MODELING OF A NOVEL TRIPLE TURBINE SOLID OXIDE FUEL CELL GAS TURBINE HYBRID ENGINE WITH A 5:1 TURNDOWN RATIO

A

THESIS

Presented to the Faculty

of the University of Alaska Fairbanks

in Partial Fulfillment of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

By

Winston Starr Burbank Jr., B.S., M.S.

Fairbanks, Alaska

August 2009

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

MODELING OF A NOVEL TRIPLE TURBINE SOLID OXIDE FUEL CELL GAS TURBINE HYBRID ENGINE WITH A 5:1 TURNDOWN RATIO

By

Winston Starr Burbank Jr.

RECOMMENDED:

Thomas Wolf

Advisory Committee Chair

Chair, Department of Mechanical Engineering

APPROVED:

Dean, School of Engineering and Mines

Dean of the Graduate School

August 10, 2009

Date

ABSTRACT

Electrical production using solid oxide fuel cell gas turbine (SOFC-GT) hybrid systems has received much attention due to high-predicted efficiencies, low pollution and the availability of natural gas. Solid oxide fuel cell (SOFC) systems and hybrid variants designed to date have had narrow operating ranges due largely to the lack of control variables available to control the thermal requirements within the SOFC.

Due to the higher value of peak power, a system able to meet fluctuating power demands while retaining high efficiencies is strongly preferable to only base load operation. This thesis presents results of a novel SOFC-GT hybrid configuration designed to operate over a 5:1 turndown ratio. The proposed system utilizes two control variables that allow the hybrid to maintain the SOFC stack exit temperature at a constant 1000°C throughout the turndown. The first control variable is the setting of a variable-geometry inlet nozzle turbine, which most directly influences the system airflow. The second control variable is an auxiliary combustor, which allows control of the thermal and power needs of the turbomachinery independently from that of the SOFC.

At low turndown the proposed hybrid operates similarly to previous hybrids, in that roughly 80% of the power is delivered from the SOFC. However, the newly proposed hybrid uses the unique turbomachinery to drastically increase the delivered power at higher power demands. A unique aspect of the proposed hybrid is the contribution of half the rated power being supplied by the inexpensive turbomachinery with the expensive SOFC contributing the other half. This will significantly lower system capital costs compared to previous hybrid designs. The proposed hybrid has high efficiencies throughout turndown with peak efficiencies occurring at low turndown levels.

TABLE OF CONTENTS

Page

LIST OF APPENDICIES

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF EQUATIONS

Page

ACKNOWLEDGEMENTS

Most importantly, thanks to my wife Diana Burbank, for being understanding and keeping the refrigerator full, and also to my parents.

I want to thank Dr. Dennis Witmer for motivating me when needed, finding humor in most any situation, and for encouraging me to finish my research in a timely manner.

Special thanks to both Dr. Thomas Wolf and Mr. Jim Kesseli of Brayton Energy LLC for their endless help.

Thank you, Dr. Ed Bueler, for always welcoming my persistent questions with enthusiasm.

Thank you, Dr. Rorik Peterson, for introducing me to Cantera. Also, distant thanks to Dr. David Goodman, from the California Institute of Technology for having developed Cantera and making it freely available.

Thanks to my many friends and colleagues in the modeling community for their encouragement and inspiration, including Dr. Jack Brouwer, Rory Roberts, Fabian Mueller, James McClay, Lorin Humphries and many other students and faculty at the National Fuel Cell Research Center (University of California, Irvine), as well as Eric Liese and Dave Tucker from NETL in Morgantown, West Virginia.

Thanks to Frank Holcomb (US Army Engineer Research & Development Center, Construction Engineering Research Laboratory) and the Department of Defense for their support through cooperative agreement number W9132T-04-2-007.

The content of this thesis does not necessarily represent the position or policy of the government and no official endorsement should be inferred.

Chapter 4 is an article accepted into the Journal of Power Sources should be cited as follows with credit also due to Dr. Wolf of Brayton Energy, LLC, Hampton, New Hampshire for his role as an advisor.

Winston Burbank Jr, Dr. Dennis Witmer, Frank Holcomb. 2009. Model of a Novel Pressurized SOFC-GT Hybrid Engine. Journal of Power Sources: POWER-D-09-00248R1

Chapter 5 was presented and published in the ASME Fuel Cell 2009 conference and proceedings and should be cited as follows.

Winston S. Burbank Jr, Dennis Witmer, Frank Holcomb. 2009. Economic Analysis of Distributed Generation Options with Wide Turndown. ASME Fuel Cell 2009: FuelCell2009-85098

Thanks to the Department of Defense for sponsoring the work presented above through contract number W9132T-04-2-007. However the content of this thesis does not necessarily represent the position or policy of the government and no official endorsement should be inferred.

CHAPTER 1: Introduction

Electrical production using solid oxide fuel cells (SOFC) has received much attention due to high-predicted efficiencies, low pollution and the availability of natural gas. An SOFC converts roughly half the chemical energy residing in a fuel, such as natural gas, into useable electricity. However, the other half of the chemical energy is converted into heat, which must be carefully monitored within the SOFC. Heat is typically removed from the SOFC by supplying extra air, where the surplus air absorbs the heat generated in the SOFC before leaving the SOFC as exhaust. Thus, while fuel and air must always be supplied to the SOFC for electrical generation, excess air must be carefully controlled to manage the thermal requirements within the SOFC. The supply of air to the SOFC requires an electric blower/compressor, which creates an electric parasitic load on the SOFC. (The electric blower uses some of the electricity produced by the SOFC).

A synergistic relationship is found when the SOFC air is supplied through the use of a gas turbine (compressor and turbine system), and the heated exhaust leaving the SOFC powers the gas turbine. This pairing of an SOFC with a gas turbine is called a solid oxide fuel cell/gas turbine (SOFC-GT) hybrid. It results in very high electrical efficiencies. Much research is under way to explore new and different SOFC-GT configurations. However, previously studied systems have had narrow operating ranges, due to an off-design mismatch between the cooling needs of the SOFC and the air supplied by the gas turbine. This discrepancy is largely attributed to the lack of system control variables available to modulate the system airflow to the thermal requirements within the SOFC.

A wide range of operation (turndown) is important because the typical electric load changes with the season and throughout any given day (as discussed in chapter 5 and represented in Figure 5.4 and Figure 5.5, respectively), largely impacted by heating and air conditioning loads, and by the use of consumer appliances. It is important that new electrical generation be able to supply this changing demand for electricity. Currently much of the world's electrical generation is supplied by coal-fired power plants, nuclear

power plants, and dammed hydroelectric plants, all of which are best suited for supplying the base electrical load. However, additional high cost electrical generation must be incorporated so that the changing electric loads can be met (known as load following generation). Due to the higher value of supplying the changing electric load (known as peak power) over the base load, a system able to meet fluctuating power demands, while retaining high efficiencies, is strongly preferable over base load generation.

This study models a novel SOFC-GT hybrid configuration designed to operate over a 5:1 turndown ratio (from 100% to 20% power generation). The proposed system utilizes two control variables that allow the hybrid to maintain the SOFC stack exit temperature at a constant 1000°C throughout the turndown. The first control variable is the setting of a variable-geometry inlet nozzle turbine, which most directly influences the system airflow. The second control variable is an auxiliary combustor, which allows control of the thermal and power needs of the turbomachinery independent from that of the SOFC.

At low turndown, the proposed SOFC-GT hybrid operates similarly to previous hybrids, in that roughly 80% of the power is delivered from the SOFC. However, the newly proposed hybrid uses the unique turbomachinery to drastically increase the delivered power at higher power demands. A unique aspect of the proposed hybrid is the contribution of half the rated power being supplied by the inexpensive turbomachinery, with the expensive SOFC contributing the other half. This will significantly lower system capital costs in comparison with previous hybrid designs. The proposed SOFC-GT hybrid has high efficiencies throughout its 5:1 turndown, with peak efficiencies occurring at low turndown levels, as shown in Figure 4.3. The proposed hybrid may find applicability in distributed generation markets where the value of meeting peak power demands is high.

A literature review on high-temperature fuel cell hybrid systems is included in chapter 2, discussing first the attempts of manufacturers to build hybrid systems and secondly, the modeling efforts being made around the world to advance our

understanding of hybrid systems. A general summary of fuel cell technology is included in Appendix A.

An overview of the model developed at UAF (and used for the results presented in this thesis) is given in chapter 3. Detailed documentation, however, is given in Appendix B, and a copy of the modeling code is provided in Appendix C.

Chapter 4 is the cornerstone of this thesis, where the proposed SOFC-GT hybrid is analyzed throughout a 5:1 turndown ratio. System efficiency and component thermodynamic profiles are presented. The proposed system offers several unique benefits over previously proposed systems.

Chapter 5 presents an economic comparison of the SOFC-GT hybrid to other distributed generation options. Analysis is made of sensitivities of variable natural gas costs compared to a fixed diesel fuel cost, as well as varied SOFC stack costs, possible carbon taxes, and variable electric costs.

Conclusions of the work presented in this thesis are made in Chapter 6.

CHAPTER 2: Literature Review

This thesis presents a novel solid oxide fuel cell/gas turbine (SOFC-GT) hybrid. The following literature review focuses on developments made by manufacturers and researchers on high-temperature fuel cell hybrids.

2.1. Current State of Solid Oxide Fuel Cells

Solid oxide fuel cells for electrical power production are commercially unavailable as of the summer of 2009. However, most manufacturers have participated in pre-commercial demonstration programs, most noticeably the Solid State Energy Conversion Alliance (SECA). The SECA demonstrations provide government funding to participants meeting performance goals, such as operating longevity, electrical availability, and decreases in manufacturing costs. However, careful inspection of fuel cell companies' financial records shows a steady decrease in capital, indicating that fuel cell companies incur financial losses each time a fuel cell is "sold" for demonstration purposes.

Fuel cells have shown little progress in entering the consumer market, and manufacturers' are even more reluctant to risk the capital necessary to build more advanced hybrid systems, unless substantial public funding is available. Siemens has demonstrated a pressurized SOFC hybrid at the University of California, Irvine, which is discussed further in section 2.1.1. FuelCell Energy also demonstrated a pressurized hybrid, however the performance details have not been made public. The lack of physical hybrids has steered researchers to build more advanced models, as discussed in section 2.3. Discussed below are the manufacturing efforts of the most predominate manufacturers.

2.1.1. Siemens Pressurized SOFC Hybrid

Siemens claims to be leading the way in SOFC-GT hybrid technology [1]. The PH200 system was a 220 kW pressurized hybrid (PH) system using a tubular SOFC stack geometry. The turbomachinery used a separated compressor and turbine (not connected by one shaft), each individually controlled, which allows separate rotation

speeds. In May 2000, the PH200 was delivered and demonstrated at Southern California Edison's National Fuel Cell Research Center (NFCRC) (located at the University of California, Irvine). As quoted from Siemens' website [1]:

> The hybrid system included a pressurized SOFC module integrated with a microturbine/generator supplied by Ingersoll-Rand Energy Systems (formerly Northern Research and Engineering Corp.) The system had a design output of 220 kW, with 200 kW from the SOFC and 20 from the microturbine generator. This system was the proof-of-concept demonstration of the SOFC/gas turbine hybrid concept. As a proof of concept demonstration it succeeded in demonstrating the concept and the operating characteristics and parameters of pressurized hybrids. It operated for nearly 3400 hours, and achieved an electrical efficiency of ~53%. This is the highest known electrical efficiency achieved by any large-scale fuel cell system. Eventually, such SOFC/GT hybrids should be capable of electrical efficiencies of 60-70%. [1]

The Siemens PH200 operated for a little over 3,200 hours (about 19 weeks). Many unforeseen control issues and instabilities were discovered during its demonstration, mostly pertaining to off-design operation, load changes, and thermal management of the SOFC. During the demonstration, the importance of matching the airflow from the turbomachinery to the cooling needs of the SOFC became increasingly clear, for both on- and off-design. Additional lessons learned from the demonstration include the following: (taken from [2])

- Precise control of the fuel compressor to match both the pressure and fuel flow is dependent upon the supplied compressed air.

- Oil, used as lubrication in the compressor and the turbine, leaked into the air stream and damaged the SOFC.

- Uneven fuel and air distribution within the SOFC leads to uneven temperatures within the stack, resulting in increased thermal stresses, uneven cell voltages, and decreased efficiencies.

- A combustor needs to be designed for operating over a range of conditions (to hold the flame in a designated area so as not to damage other components).

- The large plenum volumes of excessive piping between components was found to instigate instabilities, resulting from gas transport delays, increased stored pressure energy, and efficiency losses caused by pressure drops and heat losses.

The Siemens PH200 demonstration came very close to meeting its designed efficiency specification. However, the multitude of difficulties encountered in the PH200 demonstration, combined with the excessive cost of building the hybrid, seem to have discouraged manufacturers from making a second attempt. In personal communications with Siemens, the author has learned that Siemens has interest in demonstrating another hybrid; however, the funding is most likely unavailable for another attempt to be made. Rolls Royce has also expressed interest in building a new hybrid, and is currently using hardware-in-the-loop modeling to simulate their proposed system (as discussed in subsection 2.1.4).

2.1.2. Siemens UAF SOFC

The University of Alaska Fairbanks (UAF) began a contract with Siemens in 2005 for demonstration of a 125kW SOFC (non-hybrid). Money for the contract came from a Congressional earmark and from British Petroleum (BP) with the fuel cell scheduled for delivered to UAF in 2007 [3]. Anticipating delivery for the fuel cell, a temporary structure was erected in Fairbanks and plumbed with a natural gas supply and electrical equipment to allow the fuel cell to supply electricity to the local grid.

UAF is under a nondisclosure agreement with Siemens about the status of the SOFC. However, the lack of press releases announcing the delivery of the fuel cell represents an important data point.

2.1.3. Siemens and FutureGen

In 2003, Siemens presented preliminary plans to build a new pressurized hybrid engine expected to produce 300kW, named the PH300. Siemens proposed a large-scale SOFC power plant configured with twenty-five modular PH300 systems designed to produce 7.5MW, and named the PH7500 [2, 4]. (This module approach to the large-scale power plants is common among fuel cell manufactures). The PH300 was to improve upon the lessons learned by the PH200 demonstration (see subsection 2.1.1). However,

details of these improvements were not provided. Auxiliary equipment would still be needed during startup and shutdown, including an auxiliary combustor, used while preheating the SOFC to operating temperatures. The PH300 system is still a paper study.

Siemens has also proposed an indirect SOFC atmospheric hybrid named the AH500, where the SOFC is placed after the final turbine stage and is operated at near-atmospheric pressures. The SOFC exhaust heat is recycled by a heat exchanger, thereby increasing the temperature of the compressed air entering the combustor, before the compressed air powers the turbine(s). The AH500 uses the primary combustor during startup and peak power demands to control the stack temperature and to provide additional power. Without the combustor adding extra heat, the AH500 is expected to produce 537kW, with 15% of the power produced by the turbine having a net electric efficiency of 54%. When the AH500 uses the combustor, 639kW is expected, with the turbine supplying 30% of the electric demand at an efficiency of 49% [2]. Following industry standards, this system also remains a paper study.

The PH7500 is Siemens' proposal to "Future-Gen," the government initiative to cleanly gasify coal into syn-gas, to fuel the highly efficient fuel cell while capturing and sequestering CO_2. "At design point, the SOFC supplies 83% of the power plant output operating at roughly 60% of SOFC peak power [5]." While Future-Gen appears promising on paper, many assumptions are made, such as the reliability and cost of SOFC components over an appropriate lifetime. Impurities from coal must be cleaned with chemical scrubbers; however, these incur significant additional costs and maintenance.

By the fall of 2006, Siemens had cancelled all hybrid projects but the PH7500 system. In 2008 the U.S. Congress canceled Future-Gen due to the high costs. However, in 2009, Congress is reconsidering funding this project.

2.1.4. Rolls Royce

Rolls Royce's fuel cell division is based in Europe, where partnerships and collaborative research are strong, such as with the Thermochemical Power Group (TPG) at the University of Genoa, Italy (as discussed in section 2.3.7). In 2003, Rolls Royce

began research on an 80kW pressurized SOFC hybrid system. Rolls Royce expressed frustration in finding existing commercial components (such as compressors, turbines, and ejectors) that matched their design constraints in their proposed SOFC hybrid. Rolls Royce turned to in-house expertise to develop both the turbomachinery and ejectors for their proposed system.

In an effort to minimize costs while increasing reliability, Rolls Royce proposes replacing the SOFC anode recycle blower/compressor with a custom-designed ejector (venturi). The recycle blower must withstand the hot temperatures of the SOFC exhaust (1000°C); the non-moving/solid-state ejector appears to simplify the design. However, designing an ejector to satisfy SOFC recycle rates at different power and airflow levels is extremely difficult. Rolls Royce's proposed hybrid system also includes a second ejector for the SOFC cathode, thus replacing the expensive heat exchanger traditionally used for preheating the cathode inlet air. Both ejectors required a significant pressure drop to recycle the large percentage of flows desired. Rolls Royce's schematics show a 30% pressure drop in the inlet flows of both the air and the fuel streams [6].

Powering the ejector pressure drop and supplying cooling air to the SOFC requires a compressor/turbine with specific requirements. Rolls Royce turned to Allison Mobile Power Systems to develop a custom designed turbocharger [7]. The turbocharger is a two-spool compressor, designed to achieve a pressure ratio of 10. A single shaft connects each compressor/turbine stage and the low-pressure shaft is connected to an electrical generator, leaving the high-pressure shaft free to respond quickly to system transients [6]. However, due to the large pressure drop required by the ejectors, little or no electrical power is expected from the turbocharger.

In 2003, Rolls Royce began assembly and testing of a 80kW test stand, but simplified testing by using only two 15kW SOFC stacks, with simulated fuel cells used for the remaining 50kW. Results showed the heat distribution and gas flow within the pressurized SOFC to be very even (not unexpected given the temperature of the system was 800°C, where radiation-dominated heat transfer). Tests of the system have been executed using a significantly oversized compressor, with a computer-controlled valve to regulate mass flow and pressure to simulate the newly designed turbocharger. Likewise,

a controllable valve regulates the pressure drop that would be experienced by the newly designed turbines. Thus, hardware-in-the-loop is used to simulate the turbomachinery.

Rolls Royce, like many other manufacturers, has plans for a megawatt-sized power plant constructed by combining their 250kW hybrids in a modular fashion [8].

2.1.5. FuelCell Energy

FuelCell Energy demonstrated an atmospheric (indirect) molten carbonate fuel cell (MCFC) gas turbine (MCFC-GT) hybrid at Billings Montana in 2006 [9, 10]. The hybrid completed over 8000 hours of field-testing with 87% availability, producing 1145 MWh [10]. The hybrid claimed a 56% lower heating value efficiency (on natural gas) as it was operated as a base load supplier, but off design operation was not demonstrated [9].

FuelCell Energy has partnered with the National Fuel Cell Research Center (NFCRC) to develop transient models and control strategies for the MCFC-GT hybrid [11] resulting in several publications [12-14].

2.1.6. Other Fuel Cell Systems

Other companies have built and proposed many different designs for fuel cells, most of which are not hybrids but consist only of the fuel cell stack and support systems (balance of plant). Both Boeing and NASA have investigated using a fuel cell hybrid as an alternative power unit (APU) aboard large airlines. Such a system would supply supplemental electricity to both the plane's electronics and the ever-increasing electrical demand from passengers both in-flight and on the ground [15, 16]. Traditionally, electrical power for airlines has been supplied by the plane's jet engines. However, efficiencies are very poor at idle conditions (not in-flight) and contribute significant *NOx* pollution in populated areas. The proposed APU is a fuel cell placed into the aft/tail section of an aircraft and is supplied by the exhausted cabin air. When in flight, the cabin air is already pressurized for passenger comfort, thus the APU may not require its own compressor. Design challenges include the cost, size, weight, reliability, and the physical stresses incurred upon landings.

2.2. The Need for Modeling

Currently fuel cells are not cost competitive compared to traditional electrical generation technologies such as coal power plants or diesel generators. Both government and industry have subsidized demonstrations of fuel cells, as units have yet to become a profitable product without the subsidies.

Solid oxide fuel cells (SOFCs) are mechanically fragile, sensitive to thermal shock and stresses, poisoned by fuel impurities, and have thus far shown difficulties in meeting transient loads. Due to these issues among others, SOFC manufacturers have been reluctant to build large and/or experimental systems demonstrating cutting edge SOFC technology, at the risk of damaging the expensive SOFC. Most demonstrations have focused on achieving longevity and electrical availability, where 10,000 hours with 95% availability is considered an achievement. (10,000 hours roughly equals 14 months.) Until manufacturing costs decrease to a reasonable level and manufacturers are willing to push the limits of their systems, it remains necessary to test new configurations and control strategies by computer simulations or models.

Modeling allows fuel cell researchers to explore new and novel system configurations, fine-tune system components, and test new control strategies without risking damage to expensive physical hardware. However, all models are limited by the assumptions made and the specific question the model aims to answer. (A model based on thermodynamics can in no way predict everything a physical system will experience, such as vibrations, wear and fatigue.) An intermediate step between modeling and building a physical system is modeling hardware-in-the-loop. This is where a component (or components) of the simulated system exists and is operated as physical hardware. Information from the operating hardware is metered and used as input into the virtual computer simulation. The simulated components then predict inputs for the physical hardware, as is discussed in section 2.3.5.

2.3. Current State of Modeling

Research groups around the world are continuing to study the performance of new fuel cell/gas turbine hybrids using custom-built models. Many models are constructed to

study the dynamic and transient performance of a system, making these models useful for testing new control strategies. Given sufficient time, transient models will converge to steady-state solutions. However, steady-state models are useful during the initial investigation of new configurations, when discovering the peak performance of the engine under different load conditions. Some models are designed to simulate components at real-time speeds in order for the computer model to interact with physical hardware (as is required in a control system); this is known as hardware-in-the-loop. Lastly, models are built to answer very specific questions, and while one model may be designed to discover the system efficiency, another model would be needed to study the economics. The following subsections review different categories of models and give a summary of the modeling efforts made by several of the leading fuel cell hybrid research groups.

2.3.1. Steady-State/Thermodynamic Modeling

Many groups around the world have developed steady-state models [17-21]. These models are capable of evaluating on- and off-design performance of a system, including the overall efficiency and many other component parameters of the system (such as thermal profile, rotation speeds, and mass flows). However, steady-state models are unable to predict system transients (such as load following, ambient temperature changes, and even how the system will start up or shut down). Steady-state models are often initially used to evaluate the feasibility of new systems, where understanding the thermodynamics of the system, on- and off-design efficiencies and peak temperatures of individual components can lead to optimization of the proposed system. The system may be economically evaluated by considering the cost of materials for system components at the operating temperatures, pressures, and mass flows predicted from the steady-state analysis.

2.3.2. Brayton Energy, LLC

Dr. Thomas Wolf of Brayton Energy, LLC has developed an extensive steady-state model within Excel® using custom-written Visual Basic code. This model, combined with his expertise in turbomachinery, has produced very insightful data for several

systems [22-24]. Of greatest importance for this work is a turbine engine proposed as a replacement for diesel bus/truck engines [24].

2.3.3. Transient/Dynamic Modeling

Transient modeling allows researchers to study dynamics (rates of change) within a system, including compressor/turbine shaft instabilities, irregularities in natural gas composition, the daily and seasonal fluctuations in ambient air temperature, humidity and barometric pressure, and, most importantly, fluctuating electrical loads. System dynamics often occur on very different time scales, such as the very fast shaft dynamics of the turbomachinery and fuel cell chemical reactions, versus the very slow thermal transients of an SOFC or the daily ambient air temperature. Transient models must choose what range of time scales to study, where dynamics occurring faster or slower than the chosen time scale are neglected by the model.

Due to the fuel cell manufacturers' unwillingness to build/demonstrate new fuel cell hybrid systems, researchers have turned toward independently built models verified by other's models. It is agreed that the modeling of heat capacities, gas transport plenums (volumes), and fuel reformation are vital to accurately predicting the transient behavior of a fuel cell hybrid. However, modeling multiple behaviors greatly increases the complexity and computational time required to solve the model. Thus, all transient models make simplifying assumptions to study certain behaviors while choosing to ignore others. Even such mundane processes as the startup and shutdown of a fuel cell hybrid have proven difficult to model, and such simulations often require weeks to reach solutions [25, 26].

2.3.4. National Fuel Cell Research Center University of California, Irvine

The National Fuel Cell Research Center (NFCRC) at the University of California, Irvine (UCI) has built a transient model primarily through the work of Dr. Rory Robert [12, 13, 27-29]. NFCRC has demonstrated several fuel cell systems at their research facility, including the Siemens Pressurized Hybrid SOFC 220 (PH220). Researchers at the NFCRC are able to compare their modeling results to the collected data from the PH220 to validate their models. The NFCRC also has a thriving graduate program that

has sponsored research into molten carbonate fuel cells (MCFC), electric load dynamics of buildings, fuel cell materials, and control strategies for several different hybrid systems [12, 30].

2.3.5. Hardware-in-the-Loop

Hardware-in-the-loop is used to bridge the gap between virtual computer simulations and testing real physical hardware. In such models, part of the proposed system is simulated on a computer, while other components physically exist and operate as though they were connected to the simulated components. The simulated hardware must be able to calculate results in a timely manner (real-time), fast enough that the simulated component(s) will behave as if it physically existed. Fuel cell/gas turbine hybrids have used hardware-in-the-loop to either simulate a fuel cell with real turbomachinery or to simulate the turbomachinery utilizing a real fuel cell. There are three groups implementing hardware-in-the-loop simulations: the National Energy Technology Laboratory (NETL) which simulates a fuel cell (SOFC or MCFC) with physical turbomachinery, Rolls Royce simulates the turbomachinery with an actual SOFC, and the University of Genoa, Italy which simulates a fuel cell within several different systems. Details of these groups are given below.

2.3.6. National Energy Technology Laboratory

The National Energy Technology Laboratory's (NETL) research includes a transient model implemented within Matlab® and a hardware-in-the-loop facility named the Hybrid Performance (Hyper) facility. Hyper is a system of physical and operating compressors, turbines, combustors, and heat exchangers connected with several large plenums (volumes), but without a functioning fuel cell, forming a model hybrid system. A plenum filled with ceramic tiles is connected between the heat exchanger and turbine on the pressurized side and is used to simulate an SOFC. Using sensors to read the inlet airflow, temperature, and pressure of the plenum, data is sent to a computer model that simulates the performance of the SOFC under then given conditions. The model then predicts both the power that would be produced by a physical SOFC, and the temperature of the outlet products. The computer model also calculates how much fuel

should be burnt within the plenum, such that the plenum outlet products are similar to the predicted SOFC exhaust. The ceramic tiles within the plenum are chosen to mimic the pressure losses and heat capacity (thermal storage) of an actual SOFC.

The Hyper facility allows testing of new control strategies without risking damage to a fuel cell, as the fuel cell is modeled on a computer. This facility has also been able to validate several transient models built by NETL, NFCRC and TPG [31-33]. NETLs own transient model has been used to study several hybrid configurations, primarily testing control strategies involving bleed and by-pass valves in an effort to control the fuel cell's temperature [26, 34, 35].

Disadvantages in the Hyper facility include the fact that while the simulated fuel cell can be used to test different fuel cells designs, the turbomachinery remains unchanged. The importance of matching the turbomachinery's airflow characteristics to the cooling needs of the fuel cell has been emphasized by many literature sources [7, 13, 26, 29, 36, 37]. This has led some researchers to comment that the Hyper facility is only characterizing a bad turbine. Additionally, the Hyper facility is not compact and has extremely large plenum volumes between components, leading to increased instabilities within the system.

2.3.7. Thermochemical Power Group at the University of Genoa, Italy

The Thermochemical Power Group (TPG) at the University of Genoa, Italy, has been modeling fuel cell hybrid systems since 1997. The research group consists of approximately twenty professors, post-docs and graduate students specializing in a broad range of fields. TPG has built a strong industry partnership with Rolls Royce, aiding in development and research of both the SOFC itself and the hybrid system. TPG has also partnered with NETL and NFCRC in several publications [29, 31].

Similar to NETL, TPG has conducted research using both transient modeling software and hardware-in-the-loop simulations. However, TPG hardware-in-the-loop installations are far more compact and more closely resemble a finished product than the NETL Hyper facility [29, 38-41]. Of particular importance is the development of a

transient modeling tool built by Alberto Traverso, named Transeo. Transeo uses Simulink® while also incorporating both Matlab® and C code [21, 29, 31, 38-40, 42-45].

TPG has studied the startup procedure for a proposed Rolls Royce pressurized SOFC hybrid [46]. Investigation of the heating of the SOFC is important for both the thermal stress of the SOFC stack and the reformation of methane into hydrogen [47]. Using Transeo, it is believed that the Rolls Royce system will be able to start up without requiring external steam or external reformers to initiate reformation. This will be accomplished using recycle ejectors to recycle both heat and combustion products back into the SOFC cathode and anode inlets, thus preheating inlet air and supplying steam for reformation [38].

CHAPTER 3: Modeling Methodology

The current work and results presented within this thesis has been developed at the University of Alaska Fairbanks (UAF) as a continuation of a Master's thesis, "Building a Toolset for Fuel Cell Turbine Hybrid Modeling" [48]. This Ph.D. thesis is focused on the triple turbine SOFC-GT hybrid engine, as shown schematically in Figure 3.1. The UAF model underwent two significant changes from the toolkit developed in the Master's thesis: first, the incorporation of the open source program Cantera, used for thermodynamic calculations, and secondly, the switch from the Simulink® interface to the Matlab® environment for solving system convergence issues.

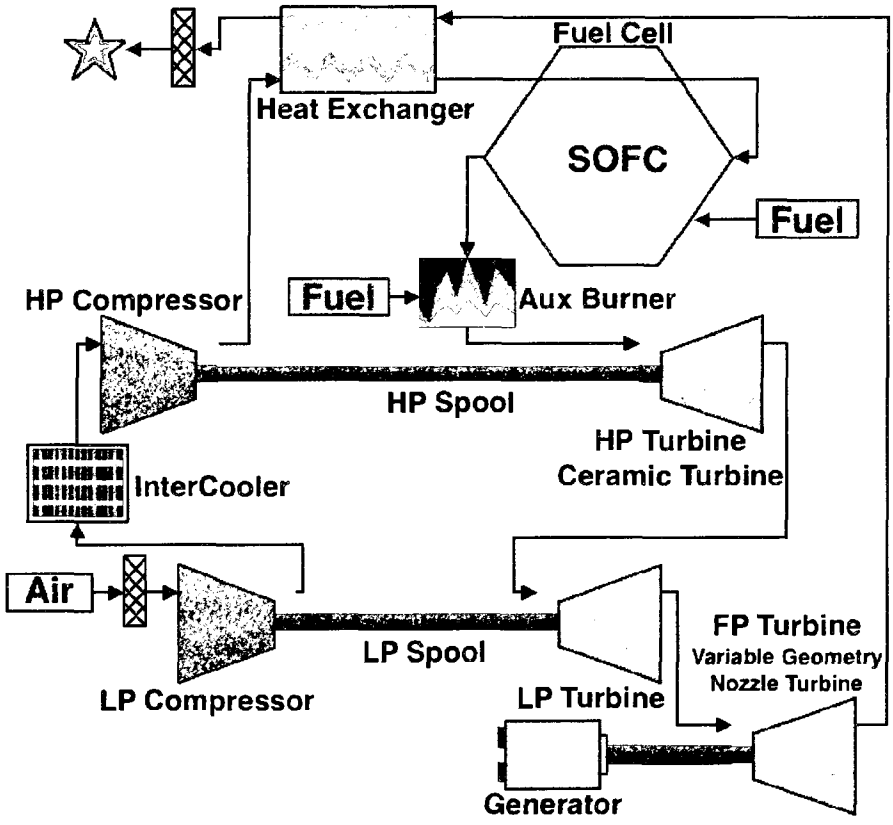


Figure 3.1: SOFC-GT Hybrid Schematic

Similar to Dr. Wolf's model (section 2.3.2), individual components are modeled where outputs of one component become inputs for the next, thus building a modular toolkit which is potentially able to model many different systems. A brief overview of

the model methodology is explained below, however, code documentation is included in Appendix C, and a copy of the Matlab® code is provided in Appendix D.

3.1. Model Validation

The UAF model has been validated by checking individual components (such as the compressor, heat exchanger, turbine, and combustor) against textbook examples found in Morgan and Shapiro's Fundamentals of Engineering Thermodynamics and Smith, Ness, and Abbott's Introduction to Chemical Engineering Thermodynamics [49, 50]. Followed by validating simple systems (consisting of one compressor, combustor, and turbine) against examples found in the same textbooks. Results of the UAF validation against textbook examples can be found in appendix E of the Master's thesis Building a Toolset for Fuel Cell Turbine Hybrid Modeling [48].

To validate more complex system configurations (those involving algebraic loops such as heat exchangers and/or connected compressors/turbine spools), the UAF model was compared to both the results produced by Brayton Energy as part of an internal study of different fuel cell hybrid configurations and to the non-hybrid ICR turbine engine [22-24, 51]. The validation of the ICR engine configuration is important because the proposed SOFC-GT hybrid is the ICR engine with the addition of an SOFC. Results between the two models differed less than 2% and where largely contributed to the differences in the design point surge margin of each compressor.

It is important to note that while the UAF model and the Brayton Energy model have many similarities, the UAF model was independently written with no code porting. The most significant difference between models is in the calculation of gas properties. The UAF model calculates gas composition properties on a molecular basis using Cantera; where the Brayton Energy model uses analytical air-to-fuel ratios. Both models calculate solutions using a multivariable solver, however the models use different variables. The UAF model solves four simultaneous variables, the low- and high-pressure compressor shaft speeds, the inlet mass flow, and the heat exchanger exit temperature, described in section 3.4. The Brayton Energy model solves five

simultaneous variables, the low- and high-pressure compressor shaft speeds, the low- and high-pressure compressor beta parameter, and the inlet mass flow [24].

Results of the ICR validation are shown below. Notice that the design point and compressor stretching for both the low- and high-pressure compressors does not lineup perfectly between models, as shown in Figure 3.2 and Figure 3.3. The UAF results are plotted in thinner symbols and lines than the Brayton Energy results. The results have been over laid so that the performance trends can be compared.

Results for the high-, low-, and free-pressure turbines lie directly on top of one another as shown in Figures 3.4-6.

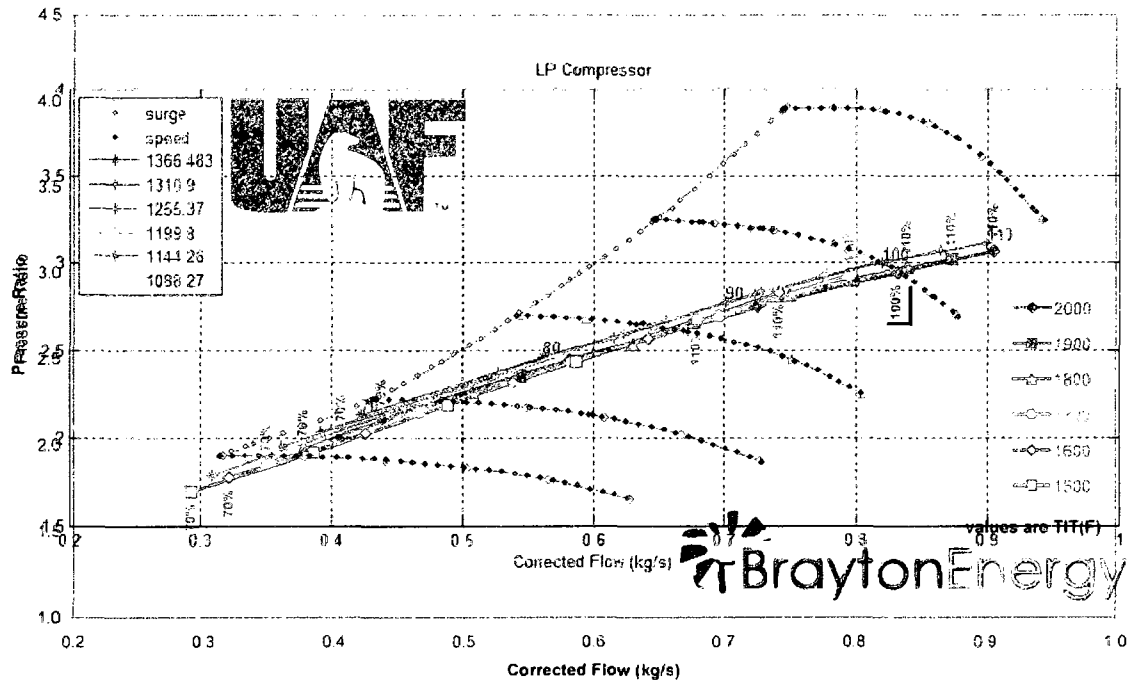


Figure 3.2: Low-Pressure Compressor Validation Results

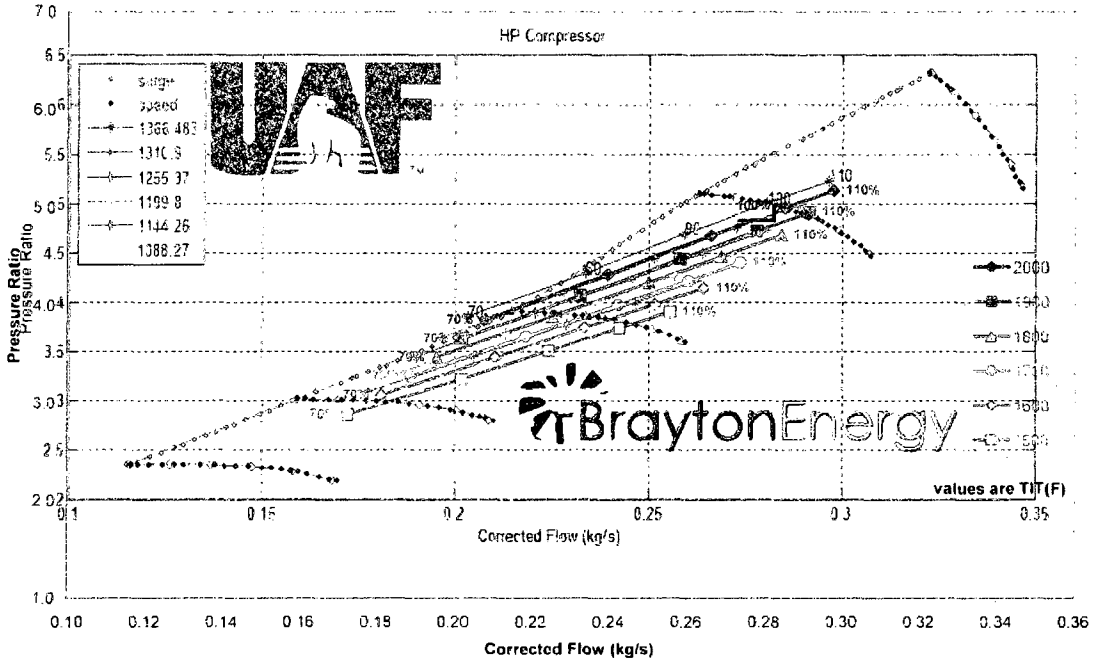


Figure 3.3: High-Pressure Compressor Validation Results

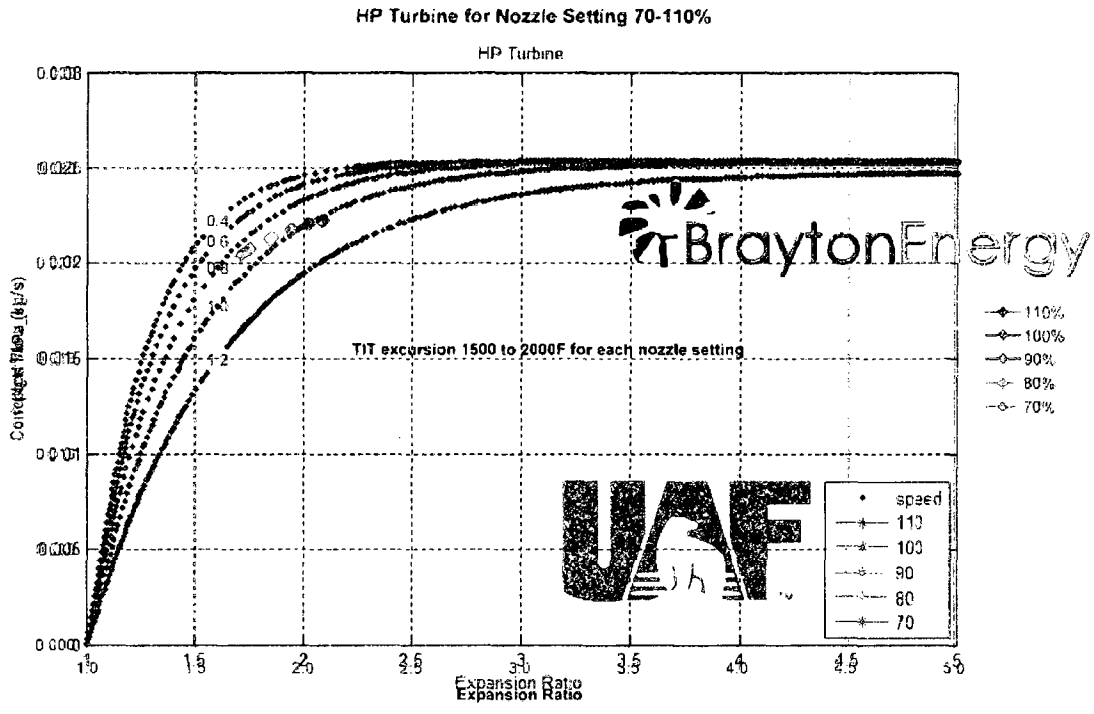


Figure 3.4: High-Pressure Turbine Validation Results

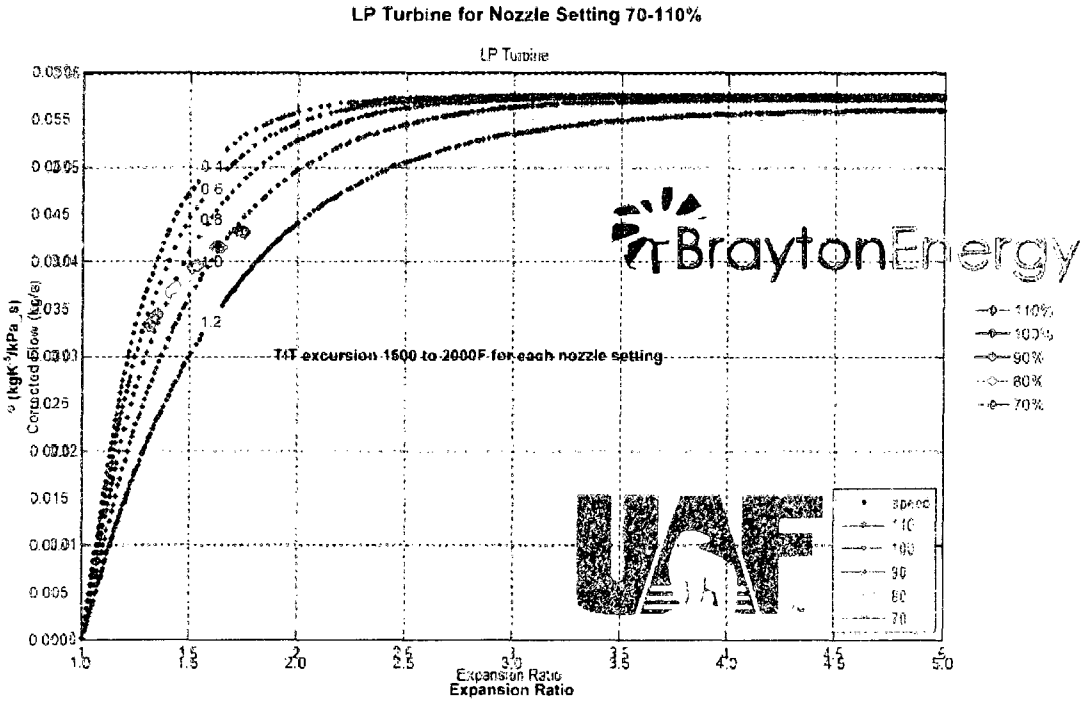


Figure 3.5: Low-Pressure Turbine Validation Results

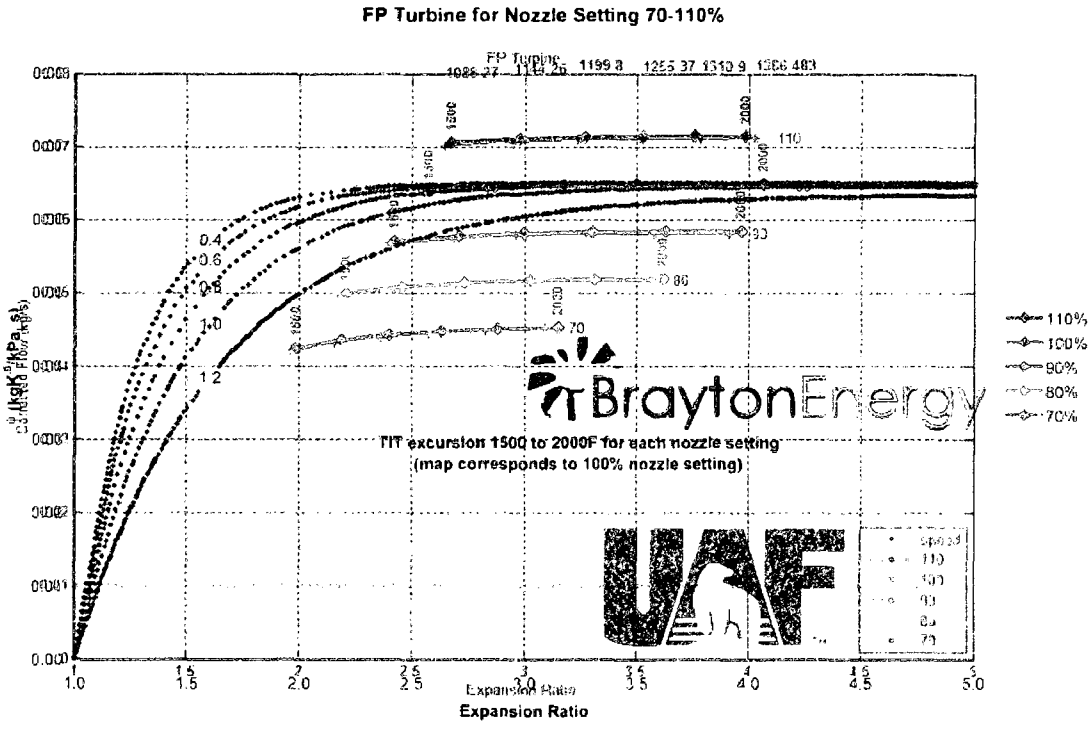


Figure 3.6: Free-Pressure Turbine Validation Results

The discrepancy in compressor design points causes a noticeable offset in the final efficiency results shown in Figure 3.7, however, the trends are identical.

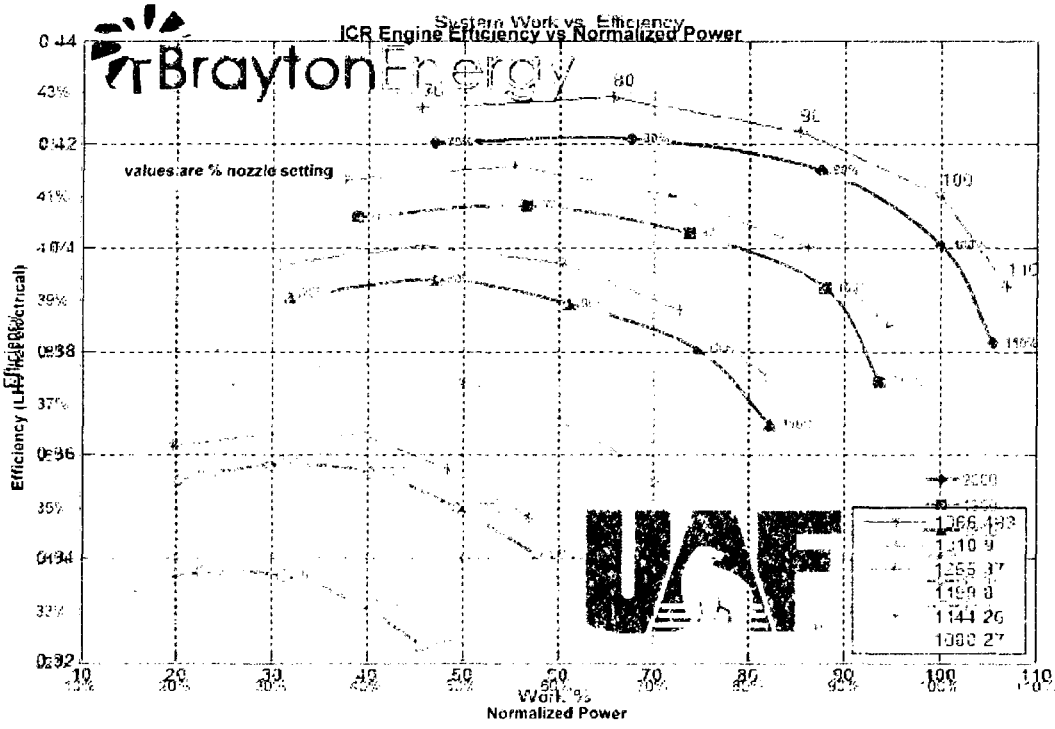


Figure 3.7: ICR System Efficiency Validation Results

The SOFC module was validated against several textbook examples. However, better validation of the SOFC module occurred with replicating the off-design performance curves presented in the Fuel Cell Handbook (Seventh Edition) section 7.2.1.1 [52].

Once validated, the UAF model calculated performance results of the SOFC-GT hybrid configuration, determining the net electrical efficiencies of the engine at different loads, as well as predicting the temperatures and pressures experienced by each component of the system, as presented in chapter 4 and published by Burbank [53].

3.2. Modeling the System

Figure 3.1 represents the proposed system studied in this work. The system airflow enters at the lower left of Figure 3.1, through a filter and into the low-pressure (LP)

compressor with a designed pressure ratio (PR) of three. The compressed air flows through an intercooler before entering the high-pressure (HP) compressor with a design PR of five. The compressed air is pre-heated by the heat exchanger (recuperator) and enters the SOFC module. The SOFC module's exhaust flows through an auxiliary burner, where the temperature can be optionally increased before entering the high-pressure (HP) turbine. The HP turbine utilizes a ceramic rotor that allows higher turbine-inlet-temperatures (TIT) than standard metal alloys. The exhaust from the HP turbine enters an un-cooled metal alloy low-pressure turbine, followed by the free-pressure (FP) turbine. The FP turbine is so named because the turbine shaft is not constrained to spin with a connected compressor, thus it is free to rotate as desired. The FP turbine utilizes variable geometry inlet nozzles that manipulate the inlet angle of the airflow, and most directly limit the choke of the FP turbine.

The schematic shown in Figure 3.1 was modeled by linking the individual model components together, such that the output from one component is used as the input for another.

3.3. On-Design and Off-Design Simulation

The simulation of the proposed hybrid is split into two modeling categories. First is on-design modeling, where the SOFC-GT hybrid uses prescribed values to determine component parameters. The second category is off-design modeling which has produced all of the performance figures presented throughout this thesis. The two categories are explained below.

A special flag is used to signal the use of the on-design modeling code. On-design modeling requires prescribed values of system components (determined by the researcher) such as the system airflow, the pressure ratios and surge margins of each compressor, temperatures and voltages within the SOFC and many others throughout the system. Full documentation of each of the components of the on-design parameters is given in Appendix C.9.1. Once the on-design simulation converges, the off-design component parameters are reported and saved. Off-design parameters include compressor and turbine performance map stretching, the SOFC cell area, the SOFC

internal heat exchanger effectiveness, and others, all of which are detailed in Appendix C.9.1.

Basically, during the on-design simulation, outputs for system components such as the compressor pressure ratios, and SOFC temperatures are prescribed. The on-design code solves certain component parameters necessary to make this happen in the off-design simulation. For example, when the SOFC cell voltage, and module and stack temperatures are prescribed, the on-design code calculates the necessary cell area such that the average current density allows the SOFC to operated at the prescribed voltage. Likewise the internal heat exchanger effectiveness is also determined such that both the SOFC cathode exit and module exit temperatures will be met in off-design simulations.

Thus, when the model is run in off-design mode at the design point condition, outputs are nearly equal to the prescribed on-design simulation values. However, now the model is calculating the component outputs from the component performance data and would do so for all off-design simulation cases. Results for the off-design simulation are within 0.0001% from the prescribed on-design values.

3.4. Converging the System

The model code was updated to be accessible directly from the Matlab® environment when it became clear that Simulink® was unable to converge on a solution due to the increased complexity of the proposed system. (This may be due to the fact that Simulink® is designed to solve transient systems, and the current model is a steady-state.) From within the Matlab® environment, the fsolve function was chosen to solve for the four system variables. (The fsolve function utilizes a Jacobian matrix that adjusts multiple variables such that the returned results approach zero. The Jacobian matrix is in essence a multi-dimensional Newtonian solver.)

The four independent system variables are as follows: the rotation rates for both the low-pressure and high-pressure spools, the system inlet airflow, and the outlet temperature on the pressurized (cold) side of the heat exchanger. (A spool is the connected compressor, shaft, and turbine, which must all rotate at the same rate.) The rotation rates are solved when the difference between the guessed rotation rate (used as

an input for the compressor) and the rotation rate calculated by the corresponding turbine approaches zero. The system air entering the LP compressor is guessed and compared to the calculated airflow desired by the FP turbine. And lastly, the heat exchanger compressed air outlet temperature is guessed and compared to the calculated value once the exhaust products are known and calculated by the heat exchanger code. The fsolve function converges when the differences between the guessed variables and the calculated values are less than 0.0001%.

The model then solves the system by converging these four variables over a range of different turbine-inlet-temperature (TIT) settings (controlled by the auxiliary burner) and different variable-nozzle settings on the free-pressure turbine, the dependant variables of the system.

3.5. Finding the Engine Turndown

The converged solutions of different TITs and variable-nozzle settings are interpolated such that engine properties can be evaluated at similar turndown levels. (The turndown refers to the rated level of power the engine produces, such that design point is 100% and half power is 50%.) The highest efficiency at each turndown is found, while respecting safe operating conditions. The necessary safe operating conditions require maintaining a surge margin larger than 5% on both the low-pressure and high-pressure compressors and respecting the maximum TIT on both the high-pressure ceramic turbine and the low-pressure metal turbine. These results discussed fully in chapter 4.

CHAPTER 4: SOFC-GT Hybrid Performance Throughout a 5:1 Turndown

4.1. Abstract

Solid oxide fuel cell gas turbine (SOFC-GT) hybrid systems for producing electricity have received much attention due to high-predicted efficiencies, low pollution and availability of natural gas. Due to the higher value of peak power, a system able to meet fluctuating power demands while retaining high efficiencies is strongly preferable to base load operation. SOFC systems and hybrid variants designed to date have had narrow operating ranges due largely to the necessity of heat management within the fuel cell. Such systems have a single degree of freedom controlled and limited by the fuel cell. This study will introduce a new SOFC-GT hybrid configuration designed to operate over a 5:1 turndown ratio, while maintaining the SOFC stack exit temperature at a constant 1000°C. The proposed system introduces two new degrees of freedom through the use of a variable-geometry nozzle turbine to directly influence system airflow, and an auxiliary combustor to control the thermal and power needs of the turbomachinery.

4.2. Abbreviations

ER: Expansion Ratio

PR: Pressure Ratio

SOFC: Solid Oxide Fuel Cell

SOFC-GT: Solid Oxide Fuel Cell Gas Turbine

TIT: Turbine Inlet Temperature

XTP: The inlet and outlet parameters of mass flow (kg/s), temperature (K), and pressure (atm).

4.3. Nomenclature

Co: Square Root of Two Times the Isentropic Efficiency

El: Total Moles of an Element within a Gas Mixture (mol)

el: Moles of an Element with a Gas Species (mol)

ER: Turbine Expansion Ratio

H: Total Gas Mixture Enthalpy (kJ s^{-1})

h: Specific Enthalpy (kJ kg^{-1})

k: Interpolated Value for Turbine Speed Lines

MS: Normalized Mach Speed

ploss: Pressure Loss across a Module (%)

P: Total Gas Mixture Pressure (atm)

Q: Heat Flux (kJ s^{-1})

rec: SOFC Anode Exhaust Recycle (%)

SS: Shaft Speed

SX: Shaft Speed Multiplier

T: Gas Mixture Temperature (K)

U: Average Turbine Tip Speed

W: Work Flux (kJ s^{-1})

X: Total Gas Mixture Mass Flow (kg s^{-1})

x: Species Mass Flow (kg s^{-1})

4.3.1. Subscripts

C: Compressor

cold: Cold side of the Heat Exchanger

hot: Hot side to the Heat Exchanger

i: Index of Single Gas Species

in: Flow Entering Module

out: Flow Exiting Module

ref: Reference State of 25°C at 1atm

T: Turbine

4.3.2. Greek symbols

ϕ: Corrected Mass Flow (kg s^{-1})

4.4. Background and Introduction

Fuel cells are devices that convert the chemical energy of a fuel into electrical energy (and heat energy), through an electrochemical reaction, and will generate electricity as long as fuel and oxygen are supplied. While this mechanism avoids some of the losses of heat engines, not all of the energy of the fuel is converted to electricity, and some heat is produced (up to half the energy of the fuel). In high temperature fuel cell systems (such as molten carbonate and solid oxide fuel cells (SOFC)), heat is removed through the exiting gas stream, and temperatures are managed by adjusting the inlet air flow, often to levels several times that required for a stoichiometric reaction [54]. This necessity of excess air for heat removal, supplied by a fan or compressor, becomes a significant parasitic loss on a fuel cell system, reducing its overall efficiency.

The need to supply high rates of airflow and remove heat suggests a natural mating of high temperature fuel cells and gas turbines, thus creating a fuel cell gas turbine hybrid. In a solid oxide fuel cell gas turbine (SOFC-GT) hybrid there is a synergistic effect in replacing or supplementing the combustor of a typical gas turbine engine with a high temperature fuel cell. The hybrid system is seen to have several benefits over that of a non-hybrid fuel cell system: higher electrochemical efficiency, lower parasitic losses, and additional electrical energy supplied from the turbine. (It is worth noting that most conventional diesel engines are hybrid systems, with the turbocharger providing excess air at increased pressures, resulting in better temperature management of the combustion process and higher efficiency).

Direct SOFC-GT hybrids are made by the placement of the SOFC within the turbomachinery, replacing the combustor and requiring the fuel cell to operate under pressure. An indirect SOFC-GT hybrid places the SOFC after the turbine and uses a heat exchanger to transfer heat from the SOFC exhaust to the air exiting the final stage of compression, but allows the fuel cell to operate at atmospheric pressures. A heat exchanger may also be incorporated into direct systems for improved efficiency. As the most noticeable difference in configurations is the pressure at which air is supplied to the SOFC; direct and indirect hybrids may also be referred to as pressurized and atmospheric hybrids respectively.

The concept of a fuel cell turbine hybrid has been discussed for many years (patented by Ztek in 1996 [55-57]). However, to date only a few SOFC-GT hybrid systems have been built and operated in demonstrations, the most publicized being the Siemens 220 kW pressurized SOFC, operated at the University of California at Irvine (Southern California Edison). While the 220 kW system operated successfully for 3250 hours, instabilities plagued the system when operated outside its narrow design envelope [2]. Siemens has not proposed building another pressurized hybrid, and none are currently scheduled. Their website indicates since 2002 only two fuel cells have been scheduled for deployment, in 2005 and 2006, however, both currently have unreported hours of operation [3].

Rolls Royce Fuel Cells has worked on developing a direct/pressurized SOFC-GT hybrid [6]. The Rolls Royce configuration is unique in its use of custom designed ejectors to recycle both anode and cathode exit gases without the need for high-temperature blowers/fans to accomplish anode recycling. In an effort to satisfy the airflow and cooling needs of the pressurized SOFC (without the need for additional bypasses, controls, or valves) Rolls Royce has been in the process of developing/modeling a custom two-stage/two spool turbocharger [7]. However, due to the large pressure drop across the ejectors and fuel cell, the turbomachinery will likely not contribute a significant amount of electrical power as compared to that supplied by the SOFC.

FuelCell Energy demonstrated an indirect 200kW molten carbonate fuel cell hybrid, however published performance data could not be located.

Development of SOFC-GT hybrids has been hindered by the difficulty of finding off-the-shelf equipment that satisfies the air delivery/thermal profile requirements of an SOFC and the cost and complexity of developing custom turbomachinery [29]. Systems built to date have used bleed values, cathode bypasses, and variable speed turbines to correct the discrepancy of airflow to thermal management. However, these methods carry the penalties of lower system efficiency and more complex control strategies [29].

Researchers at the National Fuel Cell Research Center (NFCRC), the National Energy Technology Laboratory (NETL), and the Thermochemical Power Group (TPG)

have published modeling results of SOFC-GT hybrids systems through individual and collaborative efforts. Direct SOFC-GT hybrid models and laboratory tests have been published [19-21, 26, 29, 31, 33, 58, 59]. Both TPG and NETL have test rigs able to model "hardware-in-the-loop" systems, thus validating computational models [31, 33, 41]. NFCRC and collaborators have published several transient studies as well as control strategies for indirect SOFC-GT hybrids and molten carbonate fuel cells (MCFC) hybrids [12, 13, 27, 28]. Several studies have shown large turndown in both direct and indirect hybrids, but decreasing the fuel cell stack temperature to achieve turndown has been required [12, 28].

This study is a steady-state model of a pressurized SOFC-GT hybrid, where the turbomachinery supplies the necessary airflow to the SOFC at increased pressure and Nernst efficiency, and, in turn, the exhausted heat from the SOFC powers the turbomachinery. The proposed system is unique in the use of a variable geometry inlet vane turbine, which allows the system airflow to be controlled, thus maintaining the thermal needs of the SOFC. This allows a wide turndown ratio with high efficiency, while maintaining a constant SOFC stack exit temperature of 1000°C, without the need for bleeds or bypasses. The decision to maintain the stack exit temperature at 1000°C has been made with the hope that future transient analysis will find minimal thermal transients/gradients within the SOFC, allowing the system to respond to quickly and safely to electrical transients. The proposed system is also unique in that the turbomachinery contributes significantly more power than previous systems, actually delivering more power than the SOFC at design point [7, 12, 13, 19, 26-29, 31, 39, 40, 60-66]. Thus the total cost per kilowatt is lowered for the system. During turndown the turbomachinery continues to supply the airflow/cooling needs of the SOFC, but contributes proportionally less power, taking advantage of the high efficiency of fuel cells at low power. As a result, the proposed system's efficiency increases during turndown, while the ΔT across the SOFC stack remains below 200°C.

Due to the wide turndown of the proposed hybrid, base loads can be supplied while also meeting the higher valued peak power demands. The proposed system may be found economical in distributed generation applications of stand alone use, peak

shaving, rural and remote generation, and combined heat and power (CHP) when fuel resources are available. However, due to the proposed systems higher capital and fuel costs, compared to that of coal, nuclear, hydro, and wind power plants, the SOFC-GT hybrid will likely not be economic for standby or base load operation when grid power is available [67].

4.5. System Configuration

The proposed turbomachinery for use in the proposed system could be considered a micro-turbine cousin of the Northrop Grumman / Rolls-Royce WR-21 engine, soon to be deployed by the British Royal Navy [68-70]. A schematic of the proposed system is shown in Figure 4.1, where the SOFC is the primary addition to the WR-21 configuration placed between the heat exchanger and the burner. The goal of this study is to allow a large turndown while maintaining the SOFC stack exit temperature at 1000°C, using constant fuel utilization of 85%, and without the need for cathode bypasses and/or bleeds. This is accomplished using individual fuel controllers for both the SOFC and the auxiliary burner and incorporating two novel, yet commercially viable components [24]. First, a ceramic turbine is placed on the high-pressure spool, allowing higher turbine inlet temperatures (TIT) (controlled by the auxiliary burner). Secondly, a variable-geometry inlet nozzle turbine placed after the low-pressure spool allows an additional degree of freedom in system optimization and control. This variable-geometry nozzle turbine is referred to as the "free" turbine, as it is not attached to a compressor. The system is fueled by pure methane.

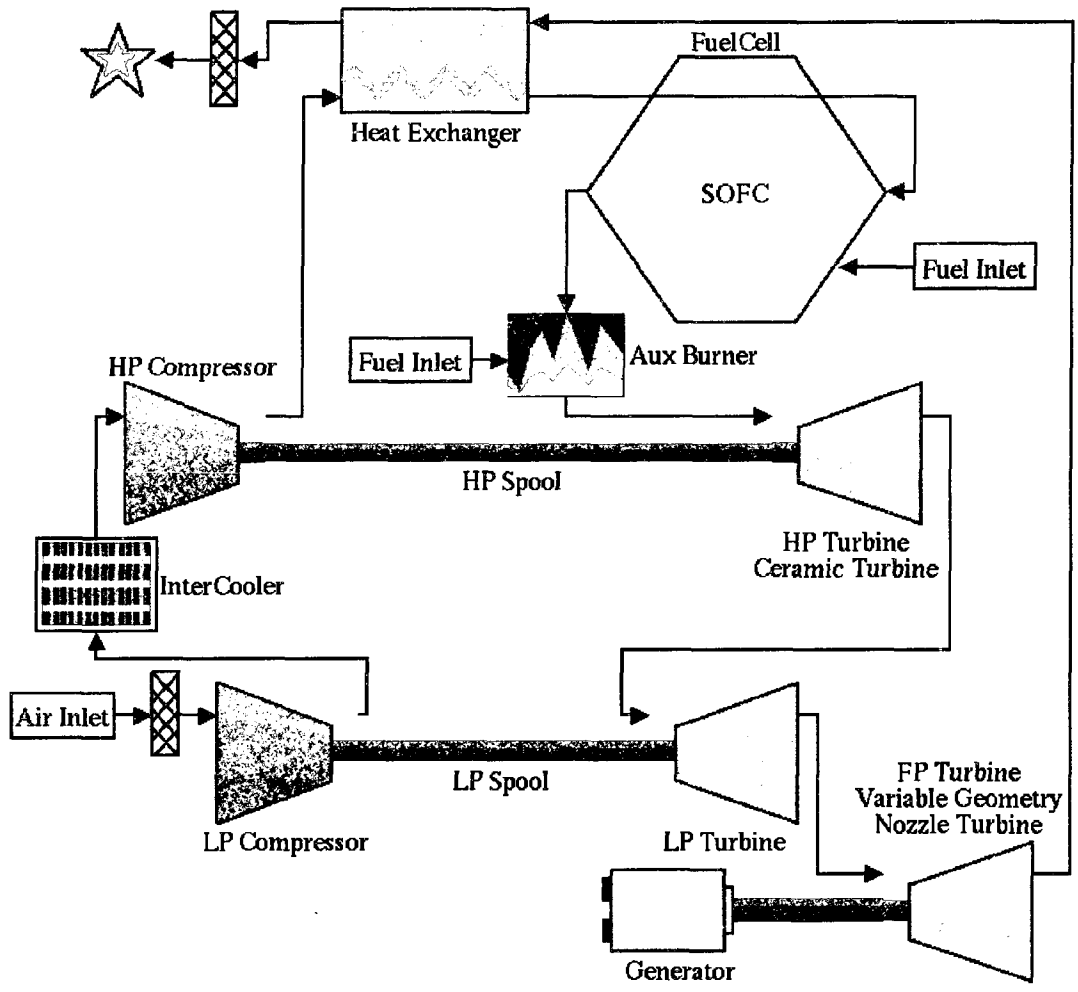


Figure 4.1: Schematic of SOFC-GT hybrid engine

The non-hybrid turbine configuration (same schematic as Figure 4.1 minus the SOFC) has been proposed as a diesel bus engine replacement, and as such, is designed so that the physical volume is quite compact [24]. The compact design minimizes the plenum volume, which is expected to decrease both transient instabilities and the amount of expensive high-temperature piping. Single shaft spools are chosen to minimize vibration concerns, and radial compressors and turbines are chosen so that inexpensive off-the-shelf turbocharger-like compressors and turbines can be utilized.

The system in this study (Figure 4.1) uses two separate single-shaft turbochargers (each similar to what can be found today on many diesel engines). These turbochargers

are compact, simple, and relatively inexpensive when mass-produced. The inlet ambient air passes through a filter into the low-pressure compressor, designed for a pressure ratio (PR) of 3. An intercooler reduces the air temperature entering the second, high-pressure compressor, designed for a PR of 5. The air is preheated by the heat exchanger and enters the SOFC module. Fuel flow into the SOFC is controlled such that the stack exit temperature remains at the prescribed level of 1000°C. The SOFC exhaust enters the auxiliary burner with its own fuel controller optionally adding additional fuel, allowing independent control of the high-pressure turbine inlet temperature (TIT). The independent control of the high-pressure TIT allows for control separation of the turbomachinery from the SOFC. The combustion products are then expanded through the high-pressure and low-pressure turbines, supplying the torque requirements to their respective/attached compressors (accounting for isentropic inefficiencies and shaft bearing losses). The exhaust then passes through the free turbine (the variable-geometry nozzle turbine) before flowing through the heat exchanger and exiting the system.

The SOFC is modeled as shown in Figure 4.2, based on a Siemens Westinghouse tubular yttria-stabilized zirconia SOFC. The inlet air is preheated by an internal heat exchanger before entering the cathode. The inlet fuel (methane) mixes with a slipstream of anode off-gases to initiate steam reformation. The anode recycle is fixed at 65% throughout this study. Nernst voltage losses are calculated from average hydrogen and oxygen concentrations, stack exit temperature, pressure, and current density as given in the Fuel Cell Handbook 7^{th} edition [52]. The anode and cathode off-gases exit at the same temperatures and are combusted before passing through the heat exchanger and exiting the SOFC.

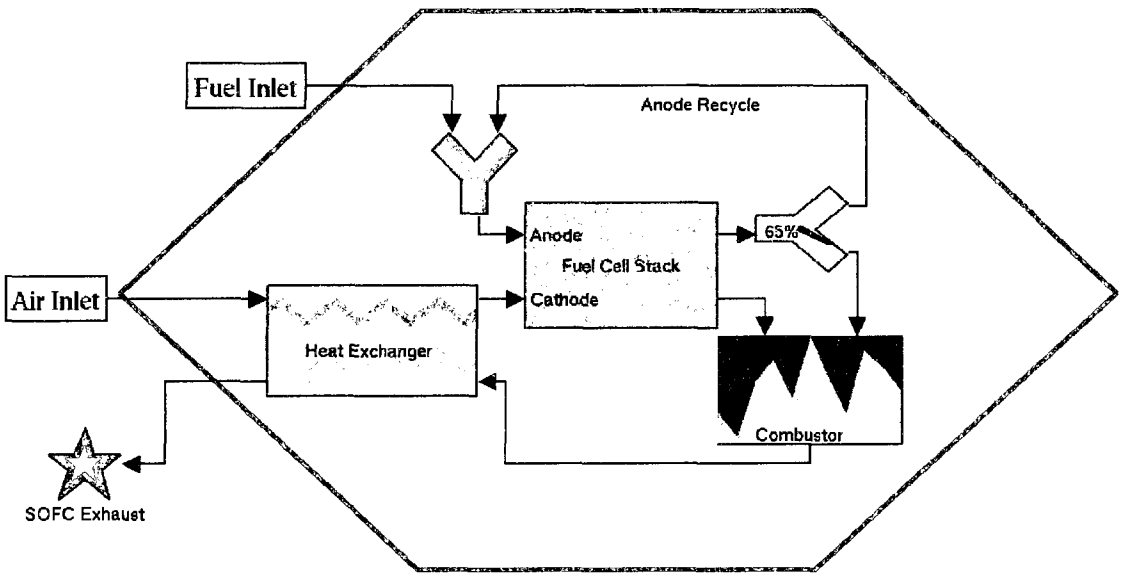


Figure 4.2: Internal schematic of the modeled SOFC

4.6. Modeling

Due to the complexity and expense of building and operating fuel cell systems, much effort has been dedicated toward computational modeling of such systems. Modeling fuel cell systems allows for exploration and development of controls that would otherwise be both expensive and difficult to achieve in a physical system. When studying a novel engine configuration, steady-state analysis at multiple states can be used to predict the efficiency and performance of the proposed system. Steady-state analysis predicts dangerous operating zones, such as excessively high temperatures, large thermal gradients, and compressor surge. Transient analysis will be needed to understand how to control the engine from one steady-state operation mode to another, and is beyond the scope of the work reported here.

4.6.1. Model Methodology

The steady-state model used in this work consists of lumped parameter components that express a balance between inlet and outlet flows of mass and energy (see Equations 4.1-4.3). Each component of the system is modeled by calculating the exiting gaseous mass flow, temperature and pressure (XTP) for use in the next component. Pressure

drops experienced across components of the system are an assumed fixed percentile loss, described by Equation 4.5. Thermodynamic properties and calculations of individual gas species and/or mixtures are accomplished through the use of Cantera, a thermodynamic toolkit developed at CalTec [71]. Cantera uses NASA polynomials of heat capacity to calculate enthalpies, entropies, and Gibbs energies. Thirteen gaseous species are considered, although more can readily be added. These are CO, CO_2, H_2, H_2O, N_2, O_2, Ar, CH_4, C_2H_6, C_3H_8, C, NO, and NO_2. In the case of electro-chemistry, reformation, or combustion, conservation of mass is verified through a calculation of conservation of elements (Equation 4.4) verifying that an equal number of carbon atoms exit as had entered.

$$H = \Sigma x_i h_i$$

Equation 4.1: Enthalpy

$$H_{out} = H_{in} + Q - W$$

Equation 4.2: Conservation of Energy

$$X_{in} = X_{out} = \Sigma x_i$$

Equation 4.3: Conservation of Mass

$$El_{in} = El_{out} = \Sigma el_i$$

Equation 4.4: Conservation of Elements

$$P_{out} = P_{in} \cdot (1 - ploss)$$

Equation 4.5: Pressure Loss

Compressor & Turbine: Turbomachinery is modeled using experimental maps for each compressor and turbine component, providing pressure and flow data at various tip mach speeds. The compressor module calculates the normalized inlet mass flow, φ (Equation 4.6), and the normalized mach speed (Equation 4.7) from the given inlet XTP (mass flow, temperature, pressure) and shaft speed parameters. Knowing the inlet φ and

mach speed, the pressure ratio (PR) and isentropic efficiency are calculated from smoothly interpolated polynomials fit to the parameterized experimental maps. Output temperature and the required work for compression are found by first assuming isentropic compression of the calculated PR, and adding the efficiency penalty to the change in enthalpy.

$$\phi_C = X_{in} \cdot \sqrt{T_{in}/T_{ref}} \Big/ \left(P_{in}/P_{ref}\right)$$

Equation 4.6: Compressor Corrected Mass Flow

$$MS = SS \cdot SX/\sqrt{T_{in}}$$

Equation 4.7: Normalized Mach Speed

The normalized turbine maps have been fitted using Equation 4.9, where intermediate speed lines are calculated by linearly interpolating k. Efficiency of the turbine is determined from a U/Co approximation [72-74]. Each compressor/turbine spool is directly linked via a rotating shaft with mechanical bearing losses equal to a percentage of the transferred power from the turbine. Within the turbine module, normalized mass flow, φ, is found from the inlet XTP parameters (Equation 4.8). The required expansion ratio (ER) is iteratively solved to meet the power requirements of the attached compressor, accounting for bearing losses and turbine efficiency. Outlet temperature is solved similarly to the compressor, assuming isentropic expansion and adding the inefficiency penalty. Knowing φ and ER, the mach speed of the turbine and the shaft speed of the spool are calculated. The system-wide solver minimizes discrepancies between the turbine shaft speed and the assumed input compressor shaft speed as is discussed in 3.4.

$$\phi_T = X_{in} \cdot \sqrt{T_{in}}/P_{in}$$

Equation 4.8: Turbine Corrected Mass Flow

$$\phi_T = 1 - e^{(-k*(ER-1))}$$

Equation 4.9: Predicted Turbine Corrected Mass Flow

Intercooler: The intercooler is placed between compressor stages and is simply modeled as decreasing the gas stream temperature to 3.9°C above that of the ambient air temperature [24]. It is assumed that a blower, moving ambient air, will be able to meet the required heat exchanger effectiveness. The parasitic load of the blower is assumed to be a constant 6.0 kW, similar to assumptions made by Brayton Energy [24].

Heat Exchanger: The primary heat exchanger/recuperator, as well as the SOFC's internal heat exchanger, are modeled using the number of transfer units (NTU) method to solve the outlet temperatures, given a prescribed effectiveness (Equation 4.10). Outlet pressures on the primary heat exchanger are found from prescribed pressure drops of 2.0% and 2.8% on the cold and hot sides, respectively, based upon the assumption made by Brayton Energy [24].

$$H_{hot\ out} - H_{hot\ in} = \Delta H = H_{cold\ out} - H_{cold\ in}$$

Equation 4.10: Heat Exchanger NTU Method

Combustor/Burner: The species output of the combustor module is found using Cantera's equilibrium function, which solves for the Gibbs energy minimization dependant upon outlet temperature, pressure, and the considered species. A prescribed heating value loss of 2.8% is assumed in the auxiliary burner, based upon the assumption made by Brayton Energy [24].

Variable-Geometry Nozzle Turbine: The free turbine is modeled similarly to the above turbine with two differences due to the variable nozzles. First, the turbine map is stretched along the corrected mass flow (φ) axis according to the percentage of the vanes' opening. Secondly, an efficiency decrement is added to the U/Co departure [75]. The outlet pressure is set at what is required for exhausting the gas, and the mach speed is found such that the turbine may operate at highest efficiency. The work produced by

the turbine is transferred to a variable speed generator, where an 8% power loss from turbine to deliverable AC is calculated, due to mechanical and electrical losses similar the assumption made by Brayton Energy [24].

SOFC: The fuel cell module (set up to model a Siemens Westinghouse yttria-stabilized zirconia (YSZ) SOFC) is modeled as depicted in Figure 4.2. Characteristics include a 65% anode recycle, allowing pre-reforming of the incoming fuel, an air pre-heater and an off-gas combustor. It is assumed that the anode and cathode streams exit at equal temperatures. Cell voltage/Nernst losses are calculated considering stack exit temperature, exit pressure, average hydrogen and oxygen concentrations, and average current density using equations and data provided on pages 7-20 through 7-31 of the Fuel Cell Handbook 7^{th} edition [52]. The SOFC module iteratively solves for the appropriate fuel flow into the anode to meet the prescribed stack exit temperature, given the fixed fuel utilization, and constant heat exchanger effectiveness. It is assumed that the anode recycle composition is equal to that of the anode exit. The normalized mass flow of the recycle stream, at steady state, is found by multiplying the anode exit by Equation 4.11.

The recycle stream mixes with the fuel inlet stream to initiate reformation. (This model does not include any energy required for the ejector or blower within the recycle stream.) An average gas concentration across the fuel cell stack is required to calculate the Nernst voltage drop on the anode side due to concentrations of hydrogen and steam. The average is calculated from the compositions of the inlet reformed fuel with the anode recycle and the anode exit.

The reformation of the inlet fuel with the anode recycled off-gas is done by modeling a steam reforming reaction followed by a water gas shift reaction that produces only H_2, CO_2, CO, H_2O, O_2, and CH_4. The anode outlet composition is calculated using a Gibbs minimization, dependant upon outlet temperature and pressure. The voltage drop due to the use of methane as opposed to pure hydrogen, at a fuel utilization of 85%, is found to be -24.1mV, consistent with numbers used by industry.

The anode and cathode streams are then adiabatically combusted as described above in the combustor module (Gibbs minimization). The products then exit the SOFC module through a heat exchanger preheating the incoming air. The fuel flow required to meet the desired stack exit temperature is calculated using the Matlab® fsolve function (a Newtonian solver). Lastly the XTP parameters of the combustion products exiting the SOFC's heat exchanger are passed out of the module, along with the net power produced assuming a 10% loss due to the DC to AC electric conversion.

$$X_{rec} = {}^{rec}\!/\!_{(1-rec)}$$

Equation 4.11: Mass Flow of Recycle Stream

Additional Assumptions:

The parasitic load of fuel compression is not considered in this study, due to the large variability in how methane can be delivered.

The effectiveness of both the stand-alone heat exchanger and the internal heat exchanger within the SOFC is assumed to be constant throughout the study.

4.6.2. System Wide Solver/Convergence

The system as depicted in Figure 4.1 is solved using Matlab's® multi-variable solver fsolve, which computes a discrete Jacobian Matrix, and iteratively varies all system variables in search of minimizing the returned vector. The returned vector consists of the difference between the guessed values and the calculated desired values of the system. The system in this study requires four variables to be solved for simultaneously. These variables are: the low-pressure spool shaft speed, the high-pressure spool shaft speed, the exit temperature from the heat exchanger on the cold/pressurized side, and lastly, the inlet air mass flow. The low-pressure and high-pressure compressor shaft speeds are compared to the desired operating speed of their corresponding turbine, having met the compressor power. Likewise, the system assumes an outlet temperature on the heat exchanger cold side. This value is then compared with the calculated value once the hot side inputs have been calculated. Lastly, the ambient air entering the system is assumed

and compared to what value is required by the free-pressure turbine (the variable-geometry nozzle turbine) to be at its peak efficiency. Using good initial values and many computational iterations, the above four variables are found such that the errors/differences approach zero (less than 0.0001% difference).

4.6.3. Design Point

The design point for the proposed system is set using multiple parameters. The primary parameters are the designed pressure ratio and surge margin of each compressor, the primary heat exchanger's effectiveness, the design point fuel cell voltage, the fuel cell stack exit temperature, the fuel cell module exit temperature (after the internal heat exchanger), the fuel cell fuel utilization and anode recycle, the high pressure turbine inlet temperature (TIT), and finally, the system airflow. Values of pressure loss are assigned to most components, and are considered constants throughout the study. The model uses specialized code to solve the system and calculates values that are then used for all off-design analysis. For each compressor a mach speed multiplier, pressure ratio (PR) multiplier, and a normalized mass flow multiplier are reported. These multipliers are used for *stretching* the compressor map and are in the range of ±7%. Each turbine reports a multiplier for normalized mass flow, shaft speed, and U (the tip speed). The SOFC reports the cell area required to satisfy the current density needed to meet the desired voltage at design point, and reports the internal heat exchanger effectiveness, which meets both the prescribed stack exit temperature and module exit temperature.

For further clarification, each compressor/turbine is *stretched* to meet the design point conditions, however once the *stretching* parameters are determined they remain fixed for all off-design modeling. This also includes the SOFC where the cell area and internal heat exchanger effectiveness are determined for the design point. However, the cell area and heat exchanger effectiveness then remain fixed and constant for all off-design modeling. The design point conditions are displayed on Table 4.1.

Table 4.1: Design Point Configuration Data

Design Point:	
TIT:	1093.35 °C
Inlet Air Flow:	1.172 kg s^{-1}
System Work:	646.704 kW
System Efficiency:	53.19%

	PR or ER	Efficiency	Inlet Temperature
LP Compressor	3	73.79%	15.00 °C
HP Compressor	5	79.75%	18.90 °C
HP Turbine	2.02	83%	1093.35 °C
LP Turbine	1.68	85%	922.38 °C
FP Turbine	3.78	85%	804.86 °C

Intercooler:	
ΔT Approach	Pressure Loss
3.9 °C	2.0%

Heat Exchanger:				
Effectiveness	Cold Side Inlet Temperature	Hot Side Inlet Temperature	Cold Side Pressure Loss	Hot Side Pressure Loss
90%	230.24 °C	550.28 °C	2.0%	2.8%

SOFC:	
Fuel Utilization	85%
Anode recycle	65%
Voltage	0.5 V
Average Current Density	720.772 mA cm^{-2}
Stack Exit Temperature	1000 °C
Modular Exit Temperature	850 °C
Heat Exchanger Effectiveness	47.48%
Cell Area	941996 cm^2
Anode Pressure Loss	5%
Cathode Pressure Loss	5%
Combustor Pressure Loss	0%
Heat Exchanger Pressure Loss	0%

Auxiliary Burner:	
Pressure Loss	Combustion Efficiency
2.80%	97.20%

Mechanical Bearing Losses:		
Low-Pressure Spool	High-Pressure Spool	Free-Pressure Turbine to AC
2%	2%	8%

Inlet Filter Pressure Loss:	0.50%
Outlet Filter Pressure Loss:	0.50%

4.7. Results

A safe and optimized turndown strategy has been found for the proposed engine after simulating more than 800 conditions of varied high-pressure TIT and variable-geometry nozzle settings. Performance graphs of the proposed engine are presented below in Figures 4.3-4.12. The black bold line with diamonds on Figure 4.5 represents the highest efficiency at varied power levels while respecting safe operating constraints. This line represents what is herein referred to as the turndown line. Safe operation constraints include: a surge margin greater than 5% on both the low-pressure and high-pressure compressors, TIT not exceeding 1093.3°C (2000°F) on the high-pressure ceramic turbine, and TIT not exceeding 927°C (1700°F) on the low-pressure turbine [24]. Diamonds are plotted on all plots in order that a correlation can be made between figures based upon the turndown percentage.

Figure 4.3 displays the system efficiency vs. rated engine power accounting for the 6 kW intercooler blower parasitic load. While the turndown line visually appears to hug the peak TIT line, the author can view, when magnified, the turndown line slowly decreasing TIT, as is shown in Figure 4.5, between the range of 95% and 70%. From the design point of 633.48 kW to a turndown of 126.7 kW the efficiency increases from 52.1% to 62.85%. This is due to the unique characteristic that at design point the power produced is evenly split between the free-pressure turbine/generator and the SOFC. As the airflow is reduced, the turbine contributes relatively less power, thus allowing the highly efficient SOFC to raise the system efficiency. Methane fuel with a lower heating value of 50026 $kJ \cdot kg^{-1}$ assumed as the fuel source.

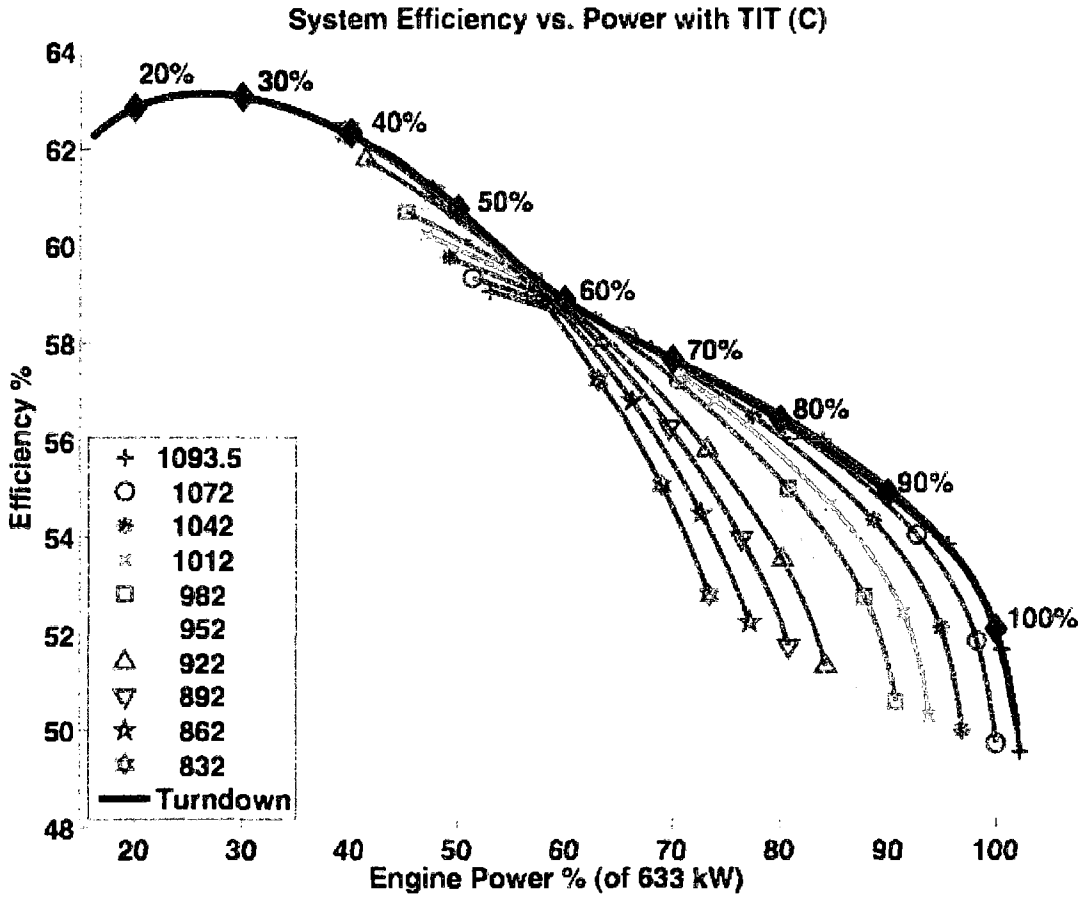


Figure 4.3: SOFC-GT system LHV electrical efficiency vs. rated system power. 100% of rated system power corresponds to 640.74 kW of produced power.

Figure 4.4 shows the power contributed by the SOFC and the variable-geometry nozzle turbine. At design point the turbine produces slightly more power than the SOFC. This shared distribution of power at design point is expected to lower system capital costs as turbomachinery typically costs between $300/kW and $500/kW, while SOFC prices are estimated at $1500/kW and above [60, 76]. Thus a significant portion of the rated power will come from the less expensive turbomachinery, resulting in a lower cost per kW of the system.

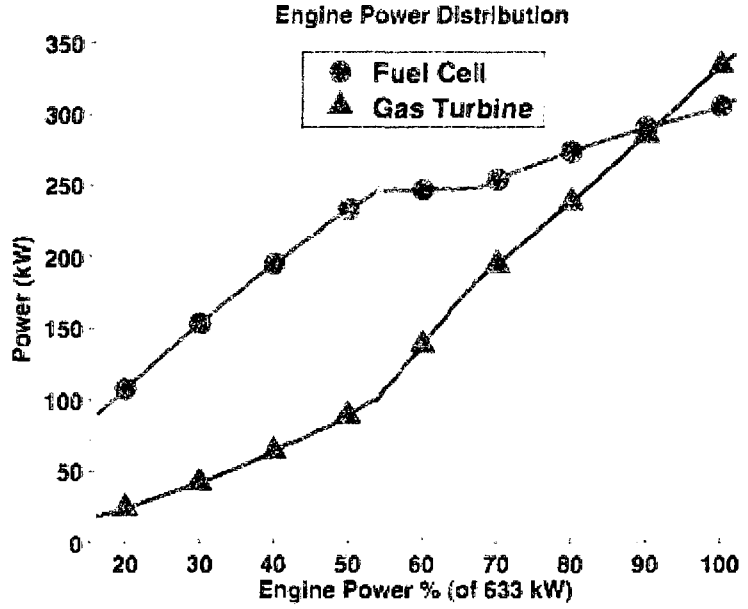


Figure 4.4: Power-sharing characteristics vs. rated power. The 6 kW intercooler blower parasitic load is not accounted for in this figure.

Figure 4.5 shows the control strategy required for the engine to follow the peak efficiency turndown, where the labeled black diamonds represent the engine's percent of rated power. Below 50% turndown, supplemental heating is not required and fuel flow into the auxiliary burner is stopped as seen in Figure 4.6. The high-pressure TIT is generated by the SOFC exhaust. Please note the TIT below 50% turndown is not the 1000°C from the stack exit, but rather the SOFC module exhaust after preheating its own inlet air. Turndown continues to 20% by reducing the variable-geometry turbine nozzle settings.

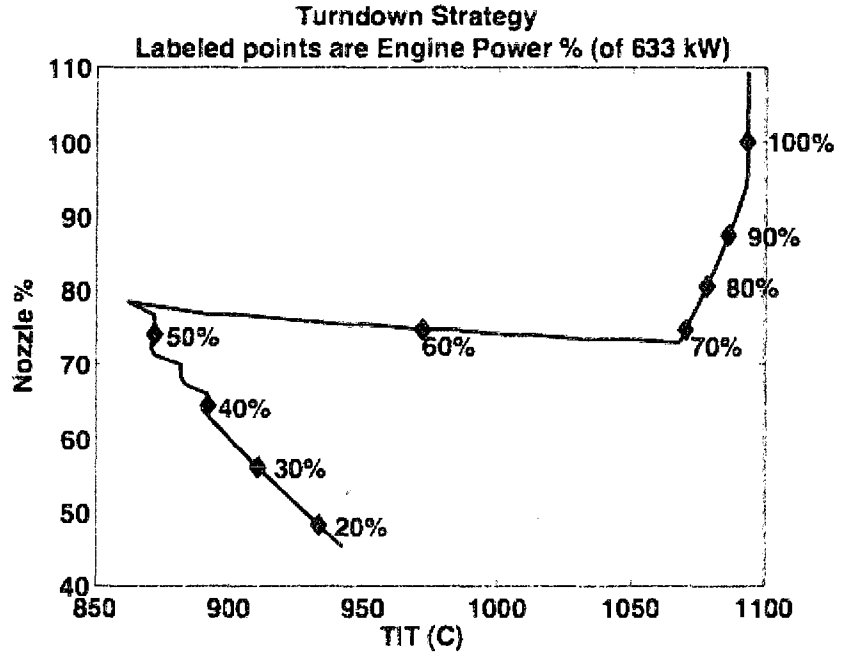


Figure 4.5: The 5:1 turndown strategy of the SOFC-GT hybrid. Labeled percentages indicate the engines rated power. The large TIT reduction experienced between 70% and 55% of the engines rated power may lead to difficult thermal transients.

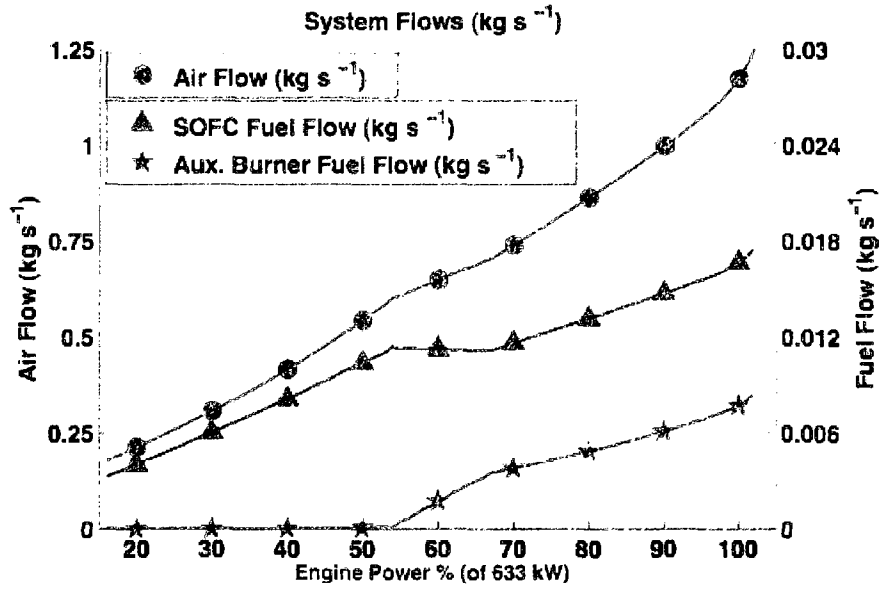


Figure 4.6: System airflow is shown as the top line with circles (left axis). Fuel flow to the SOFC is shown with triangles, while the auxiliary burner is indicated with stars (right axis).

Figures 4.7-4.9 display the operating characteristics from the low-pressure compressor, the high-pressure compressor, and the turbine spool speeds, respectively. Notice on the compressor plots, Figures 4.7 and 4.8, that an unusually large surge margin is chosen as the design point in order to maintain surge margin during turndown, which also carries an efficiency decrement. It is also important to note that compressor maps were chosen with wide and shallow surge lines for this same reason. Figures 4.7 and 4.8 as well as the omitted turbine performance maps closely resemble those published by Brayton Energy for the non-hybrid version of this engine [24]. At 20% turndown, a surge margin of 5% is held by both compressors, however if turndown continues to 16% the surge margin approaches zero, thus a 5:1 turndown is deemed safe. Figure 4.9 displays the percent deviation from the design point shaft speed for all turbine spools during turndown. The use of a variable speed generator will need to consider the 65% reduction in shaft speed experienced by the free-pressure turbine.

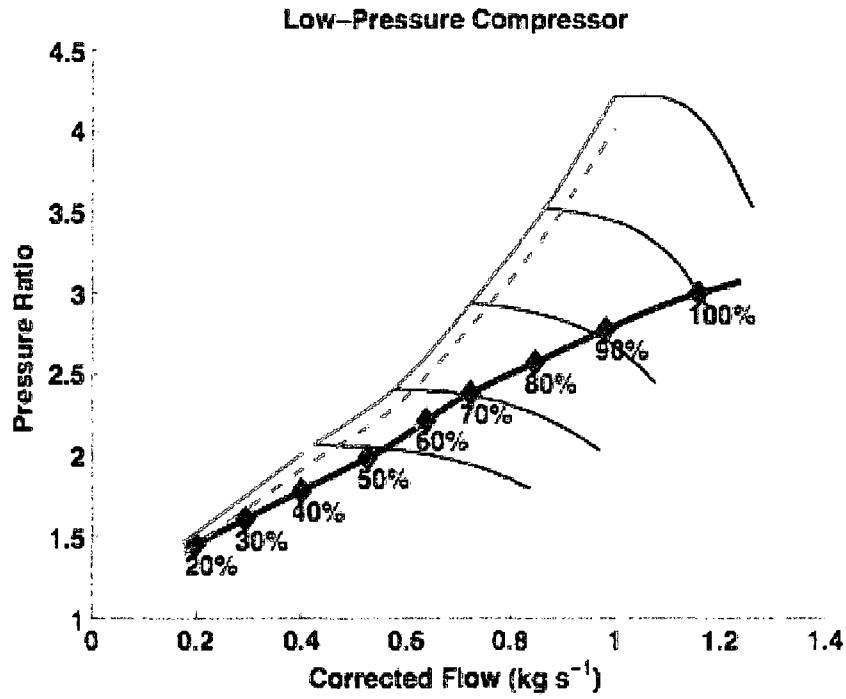


Figure 4.7: Performance of the low-pressure compressor during turndown. Labeled percentages are the engines rated power.

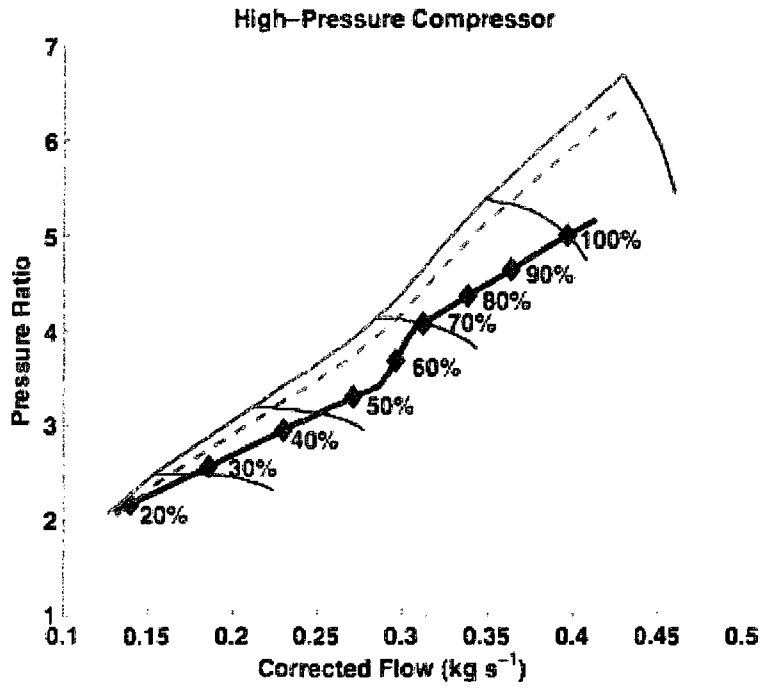


Figure 4.8: High-pressure compressor performance during turndown.

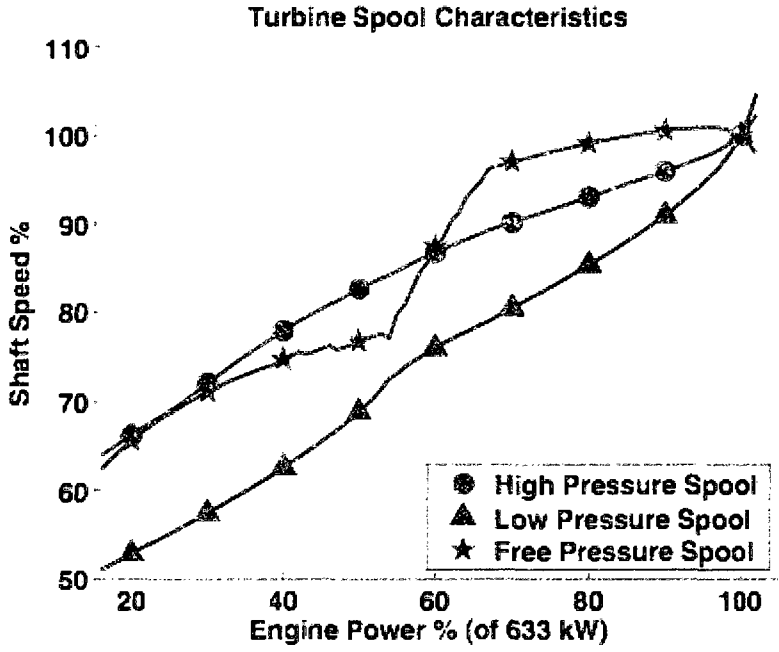


Figure 4.9: The percentage of rated shaft speed for each turbine during turndown.

Figure 4.10 shows the hot (exhaust) side temperature and the cold (inlet air) side pressure of the heat exchanger. These two parameters represent the extremes experienced by the heat exchanger. At turndown levels below 30% the temperature increases into ranges where high temperature materials may be required. Despite this increased temperature in this range the pressure differential is less than 4 atmospheres.

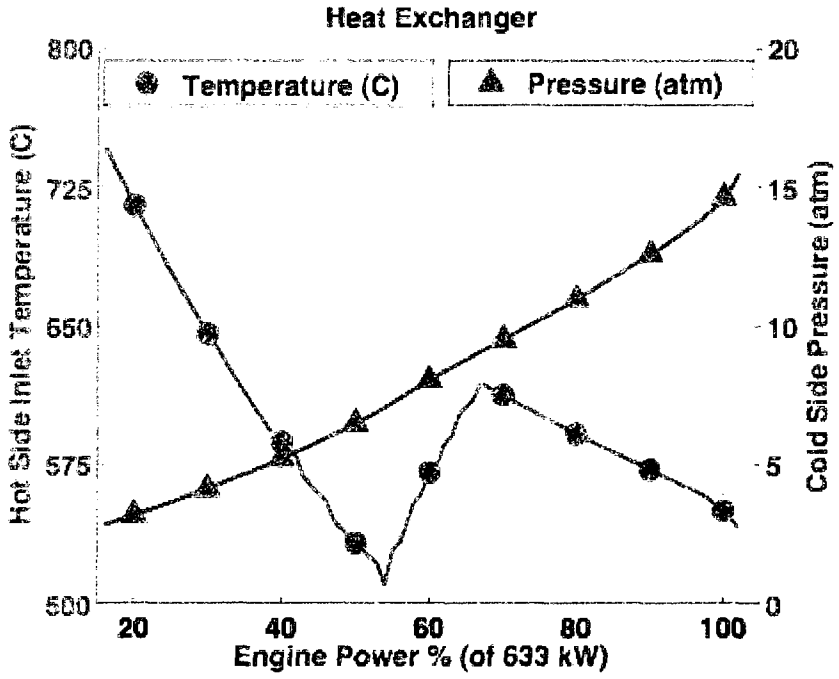


Figure 4.10: The heat exchanger extreme profiles of temperature and pressure. Temperature is from the exhaust (unpressurized and hot) side and pressure is taken from the pressurized (air inlet) side.

Figure 4.11 shows five different temperatures within the SOFC module throughout the turndown: the cathode air inlet (after preheat), the cathode exit, the post combustion of the anode and cathode streams, and lastly, the discharge from the SOFC module (after preheating the inlet air), as depicted in Figure 4.2. The proposed engine allows the cathode exit to maintain its 1000°C criterion throughout a 5:1 turndown, and limits the change in temperature across the cathode (from inlet to its outlet) to less than 200°C.

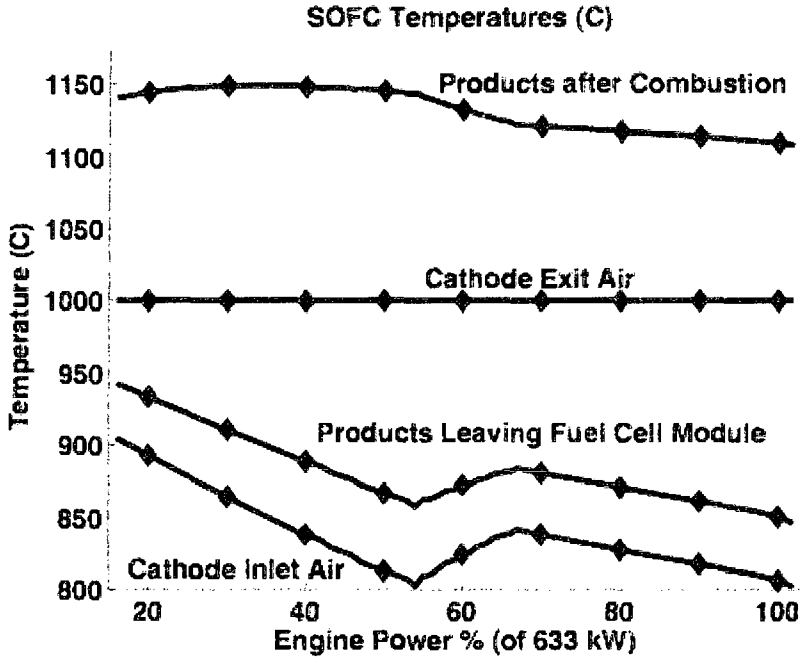


Figure 4.11: Temperature profile within the SOFC.

Figure 4.12 displays the SOFC operating voltage, average current density, and DC power output. Note both the anode and cathode of the SOFC are pressurized similarly to that experienced by the heat exchanger's cold side as shown in Figure 4.10, thus the increased pressure allows higher current densities and performance benefits within the SOFC. A fuel compressor is needed to supply methane at pressures matching that of the supplied air, however this study does not model the fuel compressor for either the SOFC or auxiliary burner. Inclusion of a fuel compressor is expected to add a significant parasitic loss to the system.

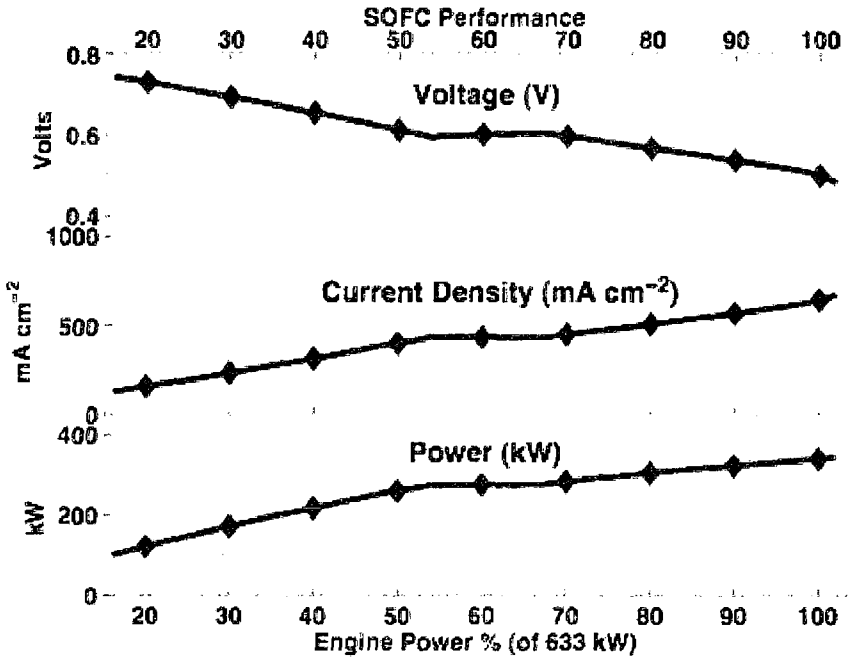


Figure 4.12: SOFC performance profiles of stack voltage, current density and power.

4.8. Discussion

The results from the steady state analysis are promising: the proposed SOFC-GT hybrid shows stable operation over a wide turndown ratio of 5:1 with remarkably high efficiencies that increase during turndown. This configuration with the ability to operate at low power levels may make startup and shutdown less stressing to the system components. The simple turbocharger-like (single shaft) compressor/turbines used for the low and high-pressure spools are expected to minimize shaft dynamics and instabilities [24]. Short plumbing interconnects are expected to minimize transient instabilities and minimize the use of high temperature materials.

Additional modeling work is necessary: a transient analysis is particularly needed between the range of 70% to 55% turndown, due to the rapid temperature gradient experienced by the hot components as shown in Figures 4.5 and 4.10. The largest thermal difference experienced by the high-pressure turbine is between 70% and 55% turndown, the TIT deceases 208°C as seen in Figure 4.5. However, due to the ability to separately control TIT (with the auxiliary burner) and system air flow (with the variable geometry turbine nozzles) there is hope the proposed system may respond to electrical

transients using the turbo machinery for quick changes while simultaneously maintaining slow thermal transients of the SOFC.

In the future, testing of the proposed hybrid may be possible through a simulated (*hardware-in-the-loop*) fuel cell, as a non-hybrid configuration of the turbine may become available as an alternative to a diesel bus engine [24]. The system capital costs will be significantly lower than that of a stand-alone fuel cell due to the nearly equal sharing of rated power between the expensive SOFC and the lower cost turbomachinery. Due to the unique characteristic of increased efficiency throughout turndown, the decision to purchase additional capacity either for future growth or for operating the engine at less than rated power for increased efficiencies could prove economical.

As mentioned in the Brayton Energy paper [24], incorporating variable inlet-guide vanes on the low-pressure compressor could increase the surge margin at low turndown and increase the compressor efficiency at design point. Increased turndown and/or startup procedures may be found if the SOFC stack temperature is allowed to decrease below the fixed 1000°C.

4.9. Conclusions

While SOFC-GT hybrids show great electrical efficiencies at design point, previous designs have narrow operating ranges. The proposed SOFC-GT hybrid shows the possibility of meeting a turndown ratio of 5:1 while retaining remarkably high efficiencies. This is done while maintaining the SOFC stack exit temperature at 1000°C, limiting the ΔT across the stack to less than 200°C, and keeping a constant fuel utilization of 85%. The proposed engine warrants additional research and should be considered a candidate for future pressurized SOFC-GT hybrid demonstrations.

4.10. Appendix: Additional Performance Figures

Below are figures that were not included with the above publication. However, they are included here for completeness.

Figures 4.13-4.15 show the performance during turndown on each of the turbines, the high-pressure turbine, the low-pressure turbine, and the free-pressure turbine

respectively. The plotted symbols represent the free-pressure turbine nozzle settings, while the thin black lines are the turbine mach speed lines.

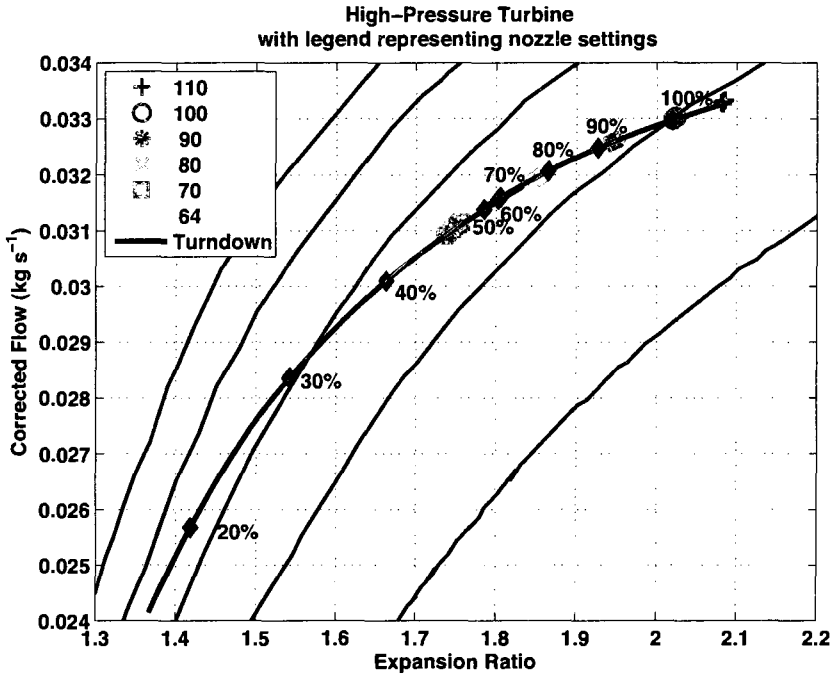


Figure 4.13: High-Pressure Turbine Turndown

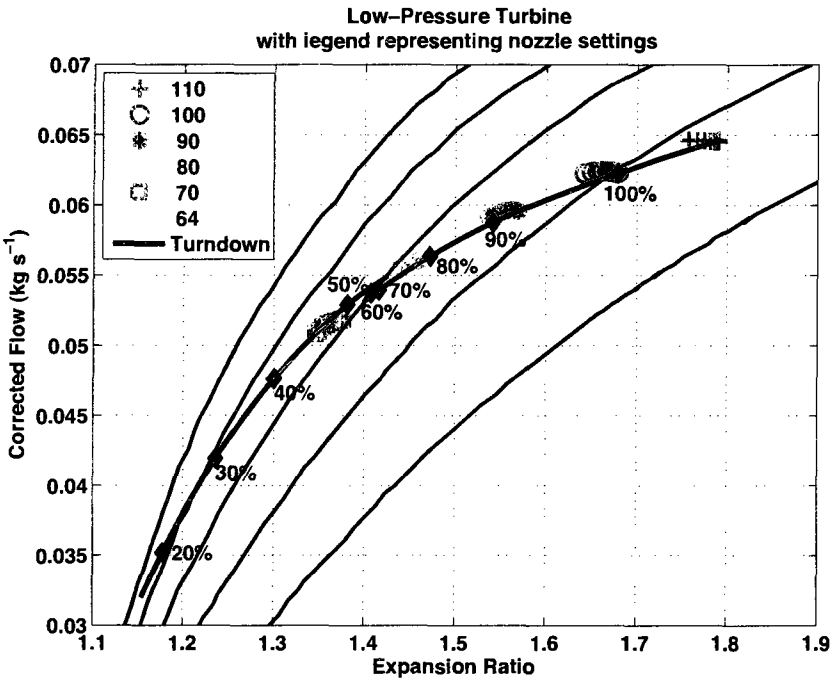


Figure 4.14: Low-Pressure Turbine Turndown

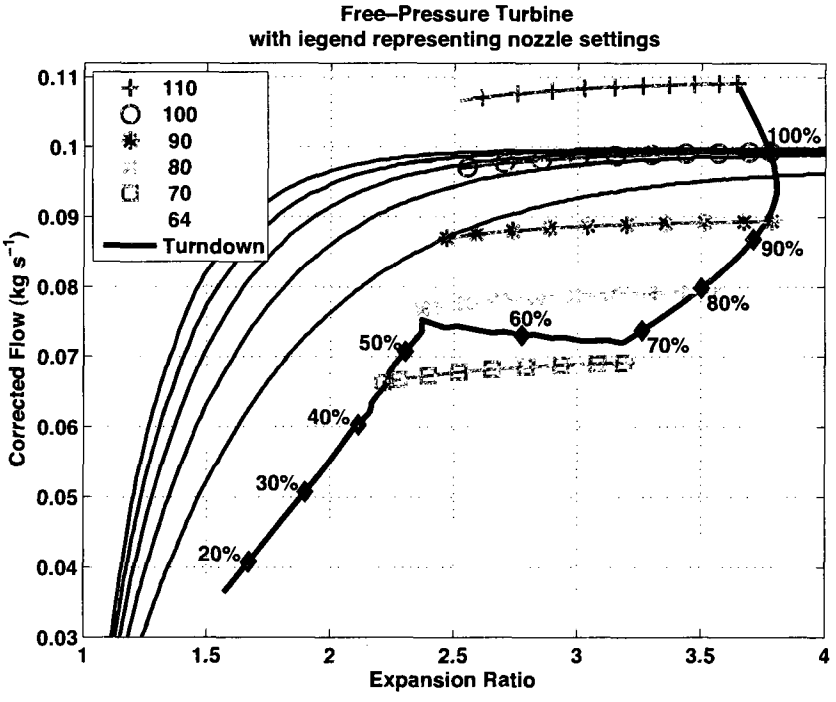


Figure 4.15: Free-Pressure Turbine Turndown

Figures 4.16-4.18 show important characteristics from each turbine during the engine turndown. On Figure 4.16 and Figure 4.17 the TIT restrictions required by the safe operating conditions can be seen. In Figure 4.16, the high-pressure turbine is limited to 1366.5K, which only occurs at the design point (100% turndown). In Figure 4.18 it is very evident that the turndown strategy is governed by the 1200K TIT restriction on the low-pressure turbine. The steep decrease in TIT experienced by all three turbines between 55% and 68% turndown is of some concern due to the rapid thermal gradient components of the system may experience. However, the overall temperature change is less than 200K. Figure 4.16 is different from the other figures in that the experienced high-pressure turbine pressure differential is shown. The high-pressure turbine utilizes a ceramic rotor and the pressure differential must be carefully considered, as the mechanical strength of the ceramic rotor is less than traditional metal alloy rotors.

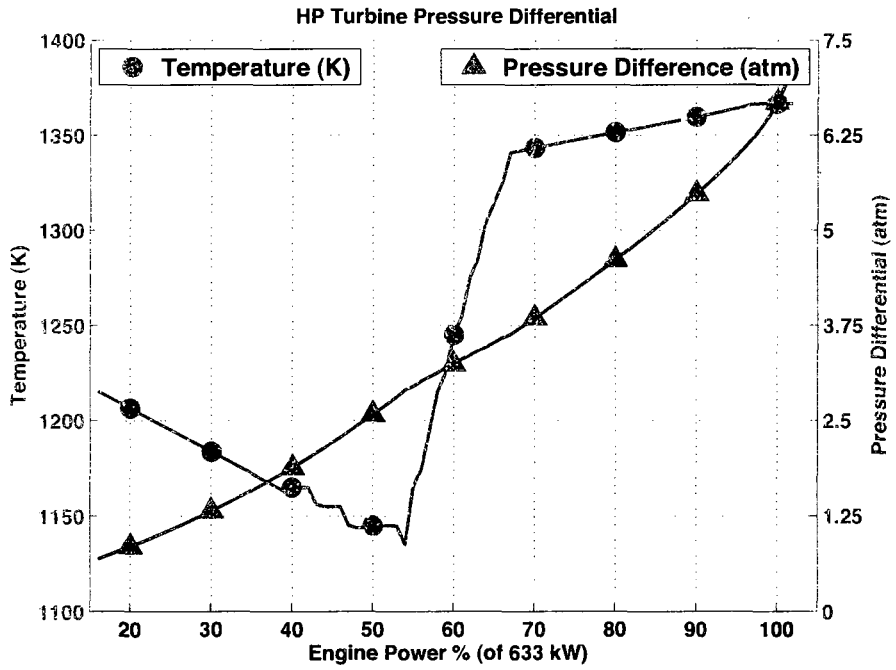


Figure 4.16: High-Pressure Turbine Temperature and Pressure Differential Turndown

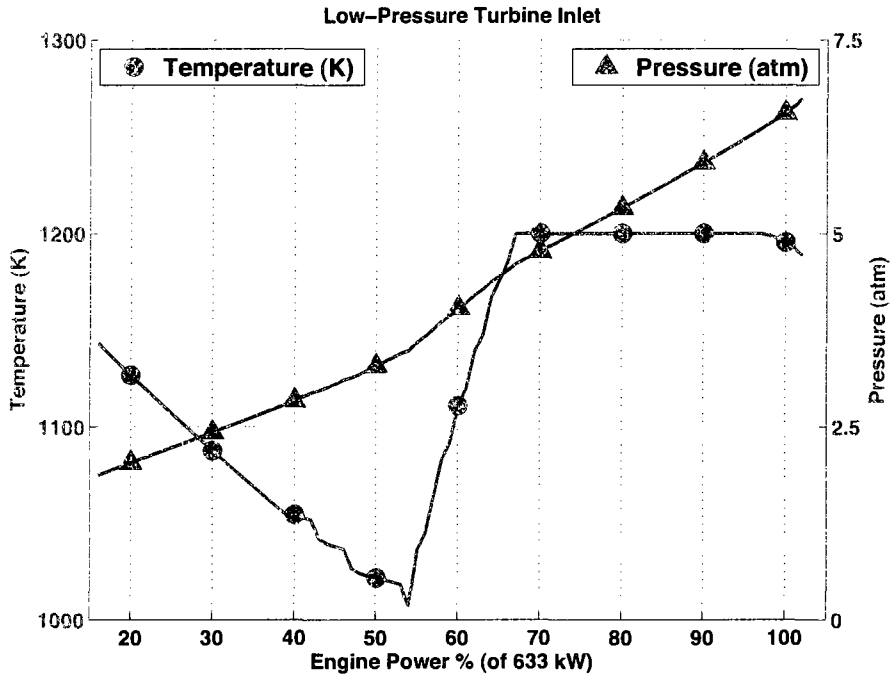


Figure 4.17: Low-Pressure Temperature and Pressure Turndown

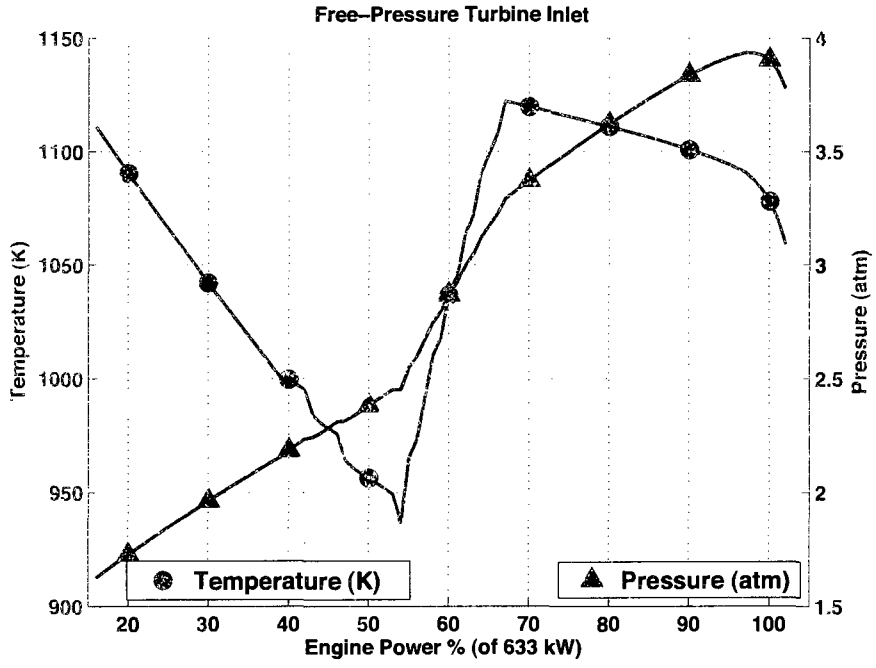


Figure 4.18: Free-Pressure Turbine Temperature and Pressure Turndown

Figure 4.19 displays both the temperature differential across the SOFC stack and the operating pressure of the stack. The current industry standard is to accept a 200°C temperature differential across the SOFC stack (from cathode inlet to outlet). The study shows the temperature differential is always less than 200°C and actually decreases during turndown.

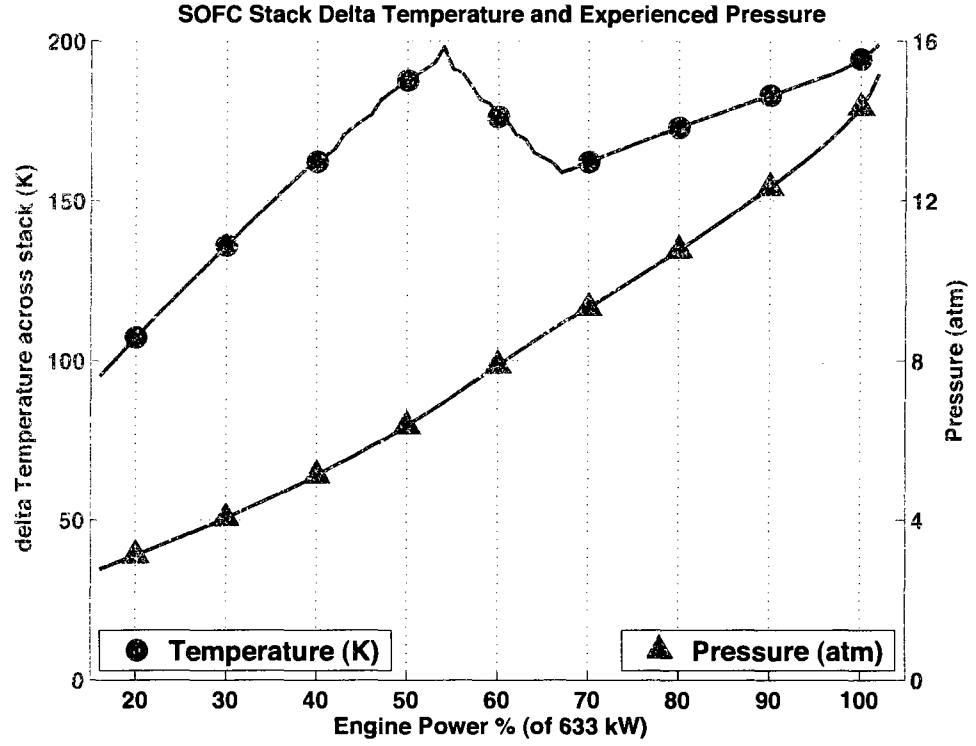


Figure 4.19: SOFC Stack Delta Temperature and Pressure Turndown

CHAPTER 5: SOFC-GT Hybrid Economic Comparison Against Distributed Generation Options

5.1. Abstract

Solid oxide fuel cell gas turbine (SOFCGT) hybrid systems have received much attention due to high predicted efficiencies, low emissions and low historical cost of natural gas. For market acceptance three criteria must be met: reliability, commercial availability and a positive net present value. This study deals primarily with the latter, comparing the net present value of the following four engines operating under a distributed or isolated loads: a simple cycle microturbine, a novel internally-cooled and recuperated (ICR) microturbine, a novel SOFCGT hybrid supported by the same ICR microturbine, and a standard diesel engine. Due to the higher value of peak power, a system able to meet fluctuating power demands while retaining high efficiencies is strongly preferable to base load operation. Sensitivity analysis is made for variable prices of natural gas, electric rates, carbon tax, and SOFC capital costs.

5.2. Introduction

Most of the developed world receives electricity from very large power plants located tens to hundreds of miles away through an electric grid. Benefits of such a system include both the reliability of generating power from multiple suppliers and competitive pricing. However, a large portion of the world does not have grid-supplied power available. For example, rural Alaska requires small, local electrical generation. Telecom and banking industries may also turn to distributed generation for increased reliability. Currently, diesel generators are the predominant choice of distributed generation due to low capital costs, predictable routine maintenance, and ease of fuel storage and use. However, gas turbines and renewable resources may be viable alternatives if the proper resources are available. Several rural Alaskan villages have known natural gas reserves; thus, alternatives to diesel generators may prove economically beneficial. In the case of a carbon tax being implemented, natural gas produces less CO_2 per kW·hr compared to diesel fuel. In this study a simple economic

net present value (NPV) is used to compare different technologies under different economic conditions.

5.3. Nomenclature

A: Electrical load amplitude (kW)

Eff: Electrical efficiency

ER: Turbine expansion ratio

i: Yearly interest rate

k: Turbine mach speed parameter

L: Electrical load (kW)

LHV: Lower heating value

NPV: Net present value

ϕ: Corrected mass flow (kg/s)

t: Time (days)

TIT: Turbine inlet temperature

5.3.1. Subscripts

$_{Avg}$: Yearly average value

$_{Day}$: Daily change

$_{Seasonal}$: Seasonal daily change

$_{yr}$: Yearly change

5.4. Discussion of Technology

In the distributed generation market there are several common alternatives: diesel generators, microturbines, and renewable resources of wind, solar, or geothermal power. This study refrains from analyzing renewable sources due to the difficulty of generalizing their performance independent of location.

As both new engines are based upon the ICR microturbine, it is important to compare the new engines to a conventional microturbine. A typical simple microturbine has a single-shaft compressor/turbine spool, uses a recuperator and is heated directly by

a combustor. The efficiency vs. rated power of a Capstone 330 MicroTurbine has been adjusted to supply a peak load of 650kW at an efficiency of 29% lower heating value (LHV) as fueled by methane gas [77]. Simple microturbines offer the advantages of quick startup and moderate noise. However, compared to other alternatives such as diesel generators, efficiencies are poor at all loads and both maintenance and capital costs are high, but fuel cost differences may give the economic advantage to the microturbine.

The Internally Cooled and Recuperated (ICR) 225 Vehicle Engine (Figure 5.1) has undergone basic design and performance analysis through a partnership between Agile Turbine Technology, Capstone MicroTurbine, and (primarily) Brayton Energy [78]. This engine consists of two single-shaft turbocharger-like-compressor/turbine spools, with an additional "power" turbine utilizing variable-geometry inlet nozzles, directly connected to a variable speed generator. The high-pressure turbine is made from ceramic, allowing turbine inlet temperatures (TIT) of 1366.5K (2000°F). However, due to current manufacturing tolerances, the ceramic turbine is limited to diameters less than 100mm, which limits the maximum power output of this configuration to 380kW [24]. The ICR 225 Vehicle Engine is sized to meet a 225kW load. However, this study assumes a configuration that has been designed for a peak power of 325kW, where two engines will be modeled to meet the peak load of 650kW. (The design is similar to the larger Northrop Grumman-Rolls Royce WR-21 designed for 21MW, scheduled for deployment in the British Royal Navy [69, 70].)

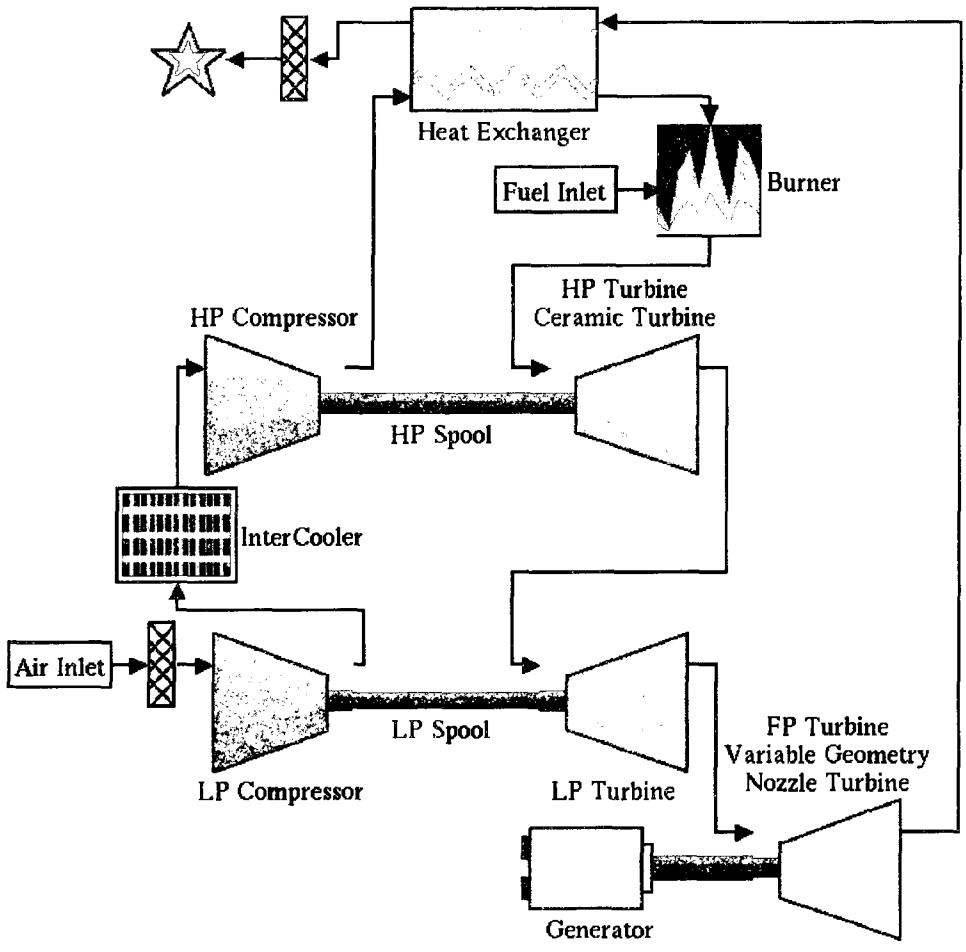


Figure 5.1: ICR 225 Vehicle Engine Schematic

The ICR 225 design is very compact and has been proposed as a replacement for diesel bus engines [24]. While power density is not as important for distributed generation applications as mobile application, compact size is expected to lower installation costs. Notable characteristics of the ICR 225 Vehicle Engine include a wide turndown ratio with relatively high efficiencies throughout (between 33% and 42%). Figure 5.3 displays the off-design efficiency extracted from reference [24]. However, efficiencies found in reference [78] are approximately 10% higher, thus this study uses the more conservative values. Maintenance of turbomachinery is low, as the number of moving components is minimal (estimated at 0.5¢/kW·hr).

The second novel engine uses the ICR 225 turbine to create a pressurized solid oxide fuel cell gas turbine (SOFCGT) hybrid shown in Figure 5.2. The turbomachinery is identical to the ICR 225 Vehicle Engine with a Siemens YSZ tubular SOFC inserted between the recuperator and combustor. Through basic design and performance modeling [23, 53], as is summarized in section 2.0, this configuration hopes to improve upon the Siemens PH200 pressurized SOFCGT hybrid operated at the University of California Irvine [2]. Potential transients are expected to be minimized due to shorter interconnect volumes. Additionally, the proposed SOFCGT hybrid retains the SOFC stack exit temperature at 1273 K while also maintaining the temperature differential across the stack to less than 200 K throughout a 5:1 turndown, potentially minimizing thermal transients experienced within the SOFC [53].

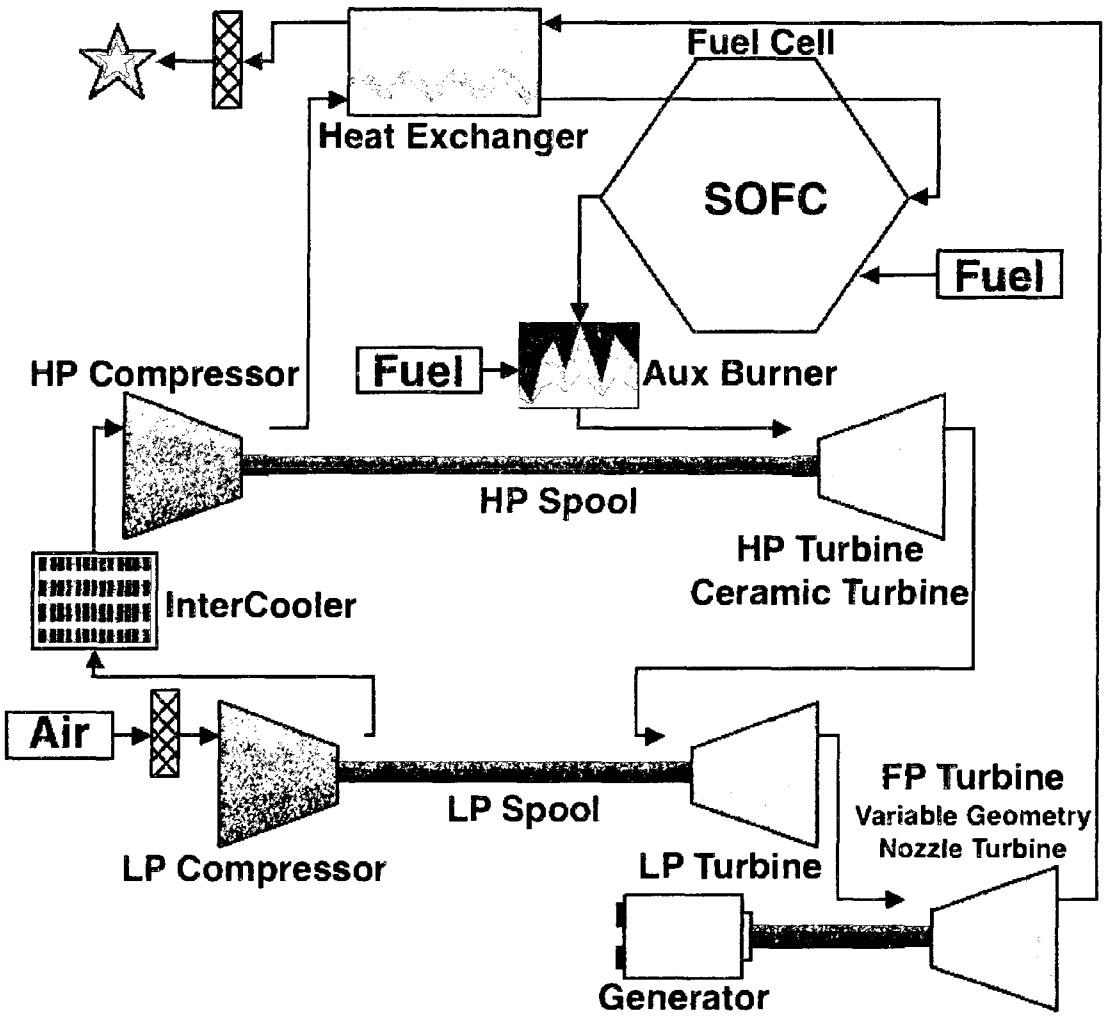


Figure 5.2: SOFCGT Hybrid Schematic

Steady state modeling shows this configuration to have the ability to meet a wide turndown ratio of 5:1, while achieving excellent net electric LHV efficiencies as shown in Figure 5.3. At design point the turbo machinery contributes slightly more power than the SOFC, resulting in a design point efficiency that is lower than other fuel cell systems. During turndown, efficiency increases as the SOFC becomes the dominant supplier of power. The sharing of peak power between the expensive SOFC and less expensive turbomachinery is expected to lower capital costs over that of previous fuel cell hybrids. The system efficiency found in reference [53] is multiplied by 90%, for use

in this study to correct for the parasitic power required by the fuel compressors as shown in Figure 5.3.

The SOFCGT hybrid is sized for 650kW peak power, roughly 325kW from each the SOFC and ICR 225 turbomachinery, limited by the size of the high-pressure ceramic turbine. Advantages include high efficiencies, low emissions, lower capital costs than fuel cell systems alone, and the hope that the rapid response of the turbomachinery can respond to transients while maintaining the thermal requirements of the SOFC. Disadvantages are the unknowns of actual SOFC costs, reliability, and performance, all of which embody the essence of commercial products.

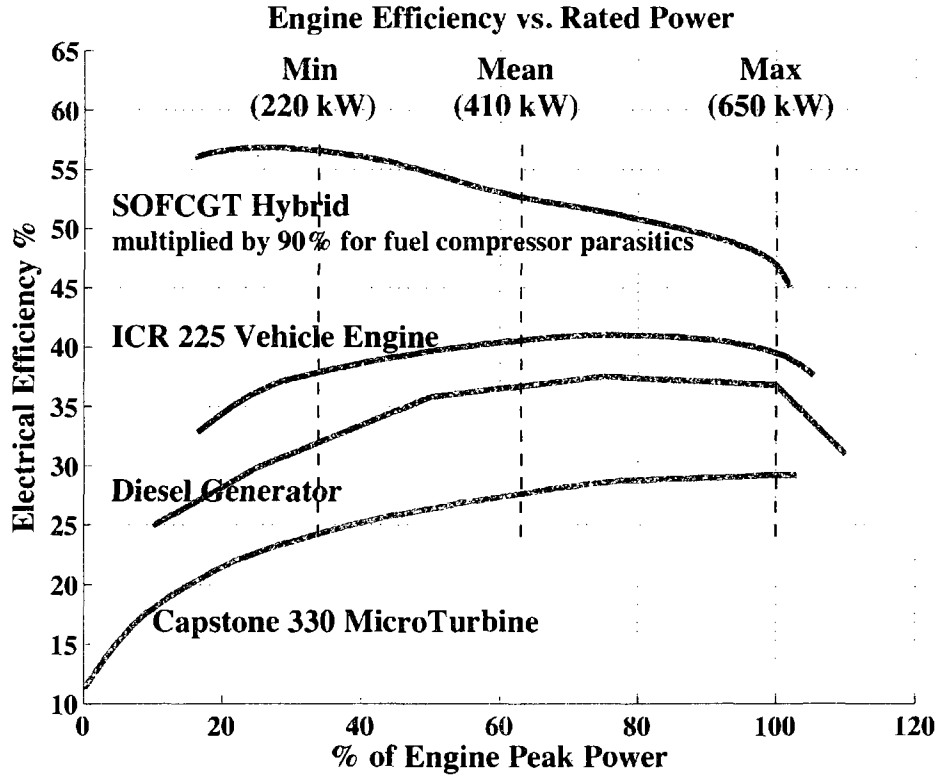


Figure 5.3: Engine Net Electrical Efficiencies LHV

Microturbines have rarely been used for distributed generation largely due to natural gas pipelines often being accompanied by grid connectivity, where distributed generation is significantly more expensive than grid power. However, natural gas may be an available resource in parts of rural Alaska and other developing regions, where

diesel fuel is currently barged or even flown in to villages, incurring significant additional expense. Additionally, the military desires power generation technology that is able to use a variety of fuels.

As the diesel generator is the current dominant technology, it is important to compare any and all alternatives to this standard. The advantages of diesel generators are the moderate capital costs, easy installation, broad understanding of the technology for maintenance, and the conveniences of transporting, storing and using a liquid fuel. The diesel generator's low capital cost is due to the decades of technological refinement and advances in mass manufacturing, and the use of only low cost readily available materials. Disadvantages include higher cost of fuel, noise, higher O&M, short lifetime, and poor off-design efficiency.

5.5. Performance Model Overview

Steady state performance modeling of the ICR 225 and the SOFCGT have been carried out through independently developed models by Brayton Energy and UAF respectively [24, 53]. Both groups modeled the ICR 225 and SOFCGT configurations with nearly identical results.

Similarities between the models include:

- Use of modular lumped parameter components that express a balance between inlet and outlet flows of mass and energy.
- Similar assumptions for bearing, pressure, generator, and combustor heat losses.
- The intercooler is modeled as reducing the pressurized air temperature to 3.9K above ambient temperatures.
- The recuperator is modeled with constant 90% effectiveness using the NTU method.
- Both models neglect the power requirements of fuel compression.
- Both models are solved using multivariable solvers.
- Brayton Energy scaled a Borg Warner automotive-turbocharger for the low-pressure compressor, used NASA's CCOD software for the high-pressure compressor,

and scaled a Borg Warner turbine map for use in all three turbines. Compressor efficiencies were found from the maps, while turbine efficiencies were evaluated in terms of u/c0 ('blade-jet-speed ratio') [75]. The UAF model digitized the fitted maps from Brayton Energy. Both models use parameterized interpolation of the compressor maps for off-design analysis. However, specifics of the parameterization were not shared. Turbine maps were fit to Equation 5.1, correlating k to mach speed lines. Intermediate speed lines are found interpolating k.

$$\phi = 1 - e^{(-k*(ER-1))}$$

Equation 5.1: Predicted Turbine Corrected Mass Flow

The models differ in the following areas:

• The Brayton Energy model is written in Visual Basic, while the UAF code is written for the Matlab/Simulink® environment.

• The Brayton Energy model uses standard bulk average gas-property correlations based on fuel-air ratios for thermodynamic calculations. The UAF model incorporates the thermodynamic software Cantera, which considers individual species of specific mixtures, using NASA heat capacity polynomials to calculate thermodynamic properties of ideal gases, and calculates a Gibbs minimization for reaction events.

• Brayton Energy assumes a constant intercooler blower parasitic load of 3.4kW, while UAF assumes 6kW.

5.6. Economic Model

This thermoeconomic model uses Equation 5.2 to simulate a yearly load fit to that predicted by the Alaska Village Load Calculator, published by the National Renewable Energy Laboratory (NREL) [79]. It is important to note that Equation 5.2 does not include random spikes and fluctuations predicted by the NREL calculator, but these random fluctuations are very important when actually sizing an engine to a village. Equation 5.2 is a sinusoidal function used to estimate the hourly load (kW), represented by L, over a given year, with t representing time in days. A_{avg} is the yearly average load,

A_{yr} is the yearly average fluctuation, A_{day} is the daily amplitude average, and $A_{seasonal}$ is the seasonal effect on daily change. For this study, A_{avg}, A_{yr} , A_{day} and $A_{seasonal}$ are 410kW, 90kW, 125kW, and 25kW respectively. The minimum, mean, and maximum loads are displayed on Figure 5.3, thus defining the operating range of each engine. Figures 5.4 and 5.5 show the yearly and daily loads (kW) respectively.

$$L = A_{avg} + A_{yr} \cdot \cos\left(\frac{2\pi t}{365}\right) - \left[A_{day} + A_{seasonal} \cdot \cos\left(\frac{2\pi t}{365}\right)\right] \cdot \sin(2\pi t)$$

Equation 5.2: Yearly Electrical Load (kW)

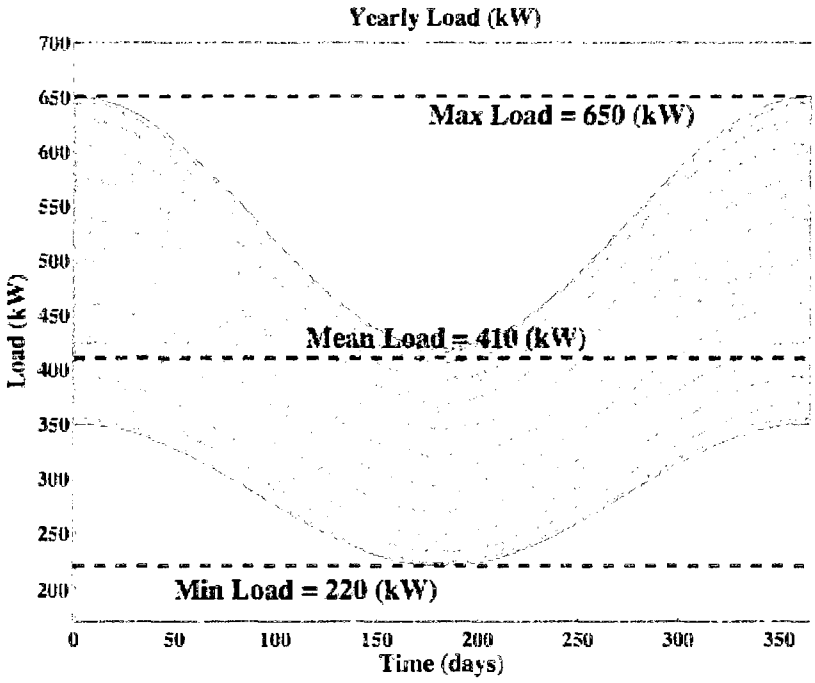


Figure 5.4: Yearly Load (kW)

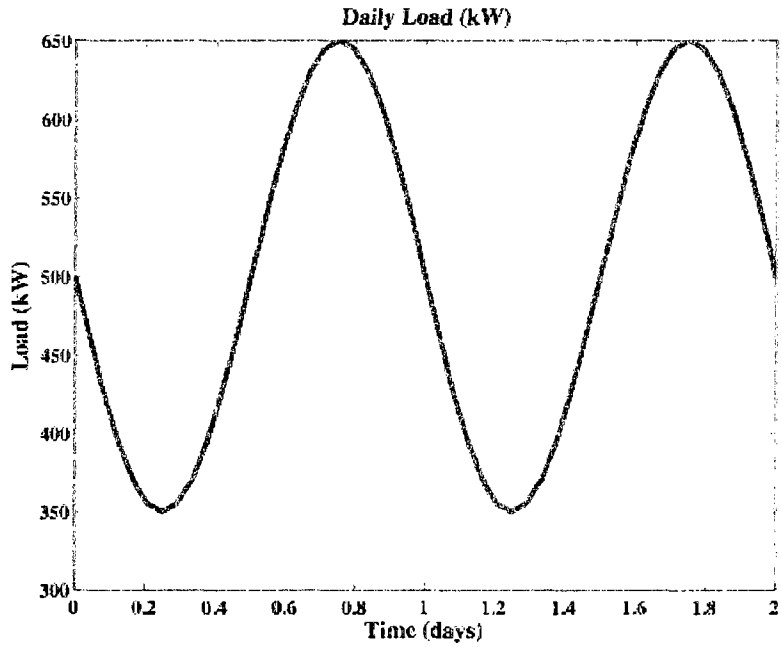


Figure 5.5: Daily Load (kW)

The yearly load profile is modeled in one-hour segments, where the load is then multiplied by one hour, yielding the work demanded (kW·hr). Each engine is assumed to produce only the demanded load, and the value of electricity is constant through each day and year.

Each engine is designed to meet a 650 kW maximum load. In the case of the ICR 225 Vehicle Engine, two 325kW engines are used in tandem; however, no optimization with respect to unbalanced loading is considered. Both the diesel generator and Capstone microturbine are assumed to meet a peak load of 650kW while retaining the efficiency curve shown in Figure 5.3. The net electrical efficiency of each engine meeting the hourly load is interpolated using the efficiency vs. power data presented in Figure 5.3. Using the lower heating values (LHV) of the appropriate fuel (as listed in Table 5.1: Fuel Properties), the total amount of fuel required to meet the demand is calculated.

Table 5.1: Fuel Properties

	Natural Gas (Methane)	Diesel
LHV	36625 MJ/1000m3 983 Btu/ft3	38658 kJ/L 0.1387 MMBtu/gal
Carbon	1967 kg CO2/1000m3 0.122812 lb CO2/ft3	2.683 kg CO2/L 22.39 lb CO2/gal
Price	$162 - $1135/1000m3 $4.67 - $32.69/MMBtu	$0.79/L $3.00/gal

The fuel costs for each hour are calculated by multiplying the amount of fuel used during each hour with the cost of each fuel. This study compares the relative difference in pricing between natural gas and diesel. Diesel costs have been fixed at $0.79/L and natural gas prices vary from $162/1000m^3 to $1135/1000m^3. (Methane at $748/1000m^3 has an equal cost per heating value as that of diesel at $0.79/L.) The operation and maintenance (O&M) costs are found by multiplying the electric demand (kW·hr) by the fixed O&M costs shown in Table 5.2. The O&M costs of the SOFCGT hybrid are assumed equal to that of the ICR, (true commercial O&M costs for fuel cells are largely unknown). The income from electricity sales is calculated by multiplying the electrical demand (kW·hr) by the variable value of electricity listed in Table 5.3.

Table 5.2: Engine Properties

	Capstone MicroTurbine	ICR 225 VE	SOFCGT Hybrid	Diesel Generator
Capital	$1000/kW	$400/kW	$950 - $2700/kW *	$800/kW
O&M	3.1¢/kW·hr	0.5¢/kW·hr	0.5¢/kW·hr	2¢/kW·hr
Lifetime	All engines are studies considering a 5-year lifetime.			
* SOFCGT Hybrid capital costs use ICR capital costs for half the rated power with the second half of power supplied by the SOFC, with costs ranging between $1500/kW to $5000/kW.				

Table 5.3: Electric Value and Carbon Tax

Electric Value	[15¢, 25¢, 35¢]/kW·hr
Carbon Tax	[0¢, 5.5¢, 11¢]/kg CO_2 [$0, $50, $100]/ton CO_2

Carbon emissions have been calculated assuming complete conversion of the fuel into H_2O and CO_2, where the fuel used is multiplied by the pounds of CO_2 per amount of fuel as listed in Table 5.1: Fuel Properties. The carbon tax for each hour is found by multiplying the pounds of CO_2 produced with the variable carbon tax (¢/kg CO_2) shown in Table 5.3.

Capital costs for each engine are shown in Table 5.2. Installation costs for each engine are assumed to be similar and neglected in this comparative analysis. A unique feature of the SOFCGT hybrid is that at design point the produced power is equally shared between the SOFC and the ICR turbomachinery, so the capital cost reflects this spilt. The SOFCGT hybrid uses $400/kW for the 325kW contributed by the ICR turbine, and values between $1500/kW and $5000/kW for the remaining 325kW supplied by the SOFC. This yields capital costs for the SOFCGT hybrid between $950/kW and $2,700/kW (for the 650kW engine total costs range between $308,750 and $877,500).

Equipment lifetimes are assumed to be five years for all engines. It is assumed that the equipment is operated continuously for its lifetime without consideration for offline maintenance. The net income is found by subtracting the carbon tax, O&M, and fuel costs from the value of sold electricity. This value is then summed for the entire year and indicated by *Net*. Net present value (NPV) is calculated using Equation 5.3, with a 6% interest rate represented by *i*.

$$NPV = \left(\sum_{Yr=1}^{Lifetime} \frac{Net}{(1+i)^{Yr}} \right) - Capital$$

Equation 5.3: Net Present Value

5.7. Results

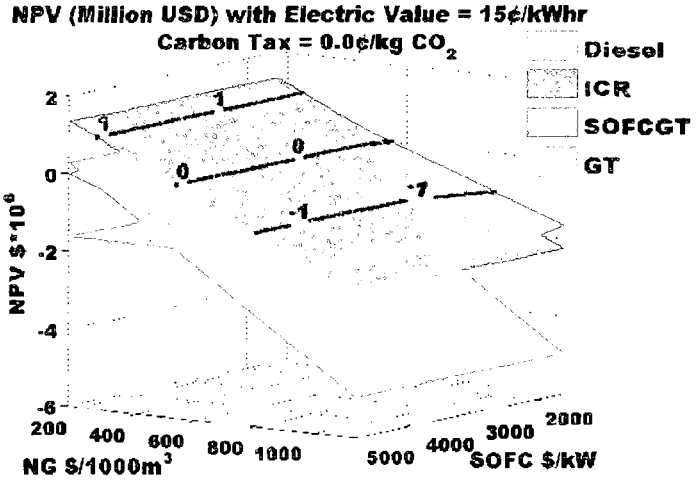
The results have been plotted as 3D surface plots with variable SOFC capital costs and natural gas costs as the x- and y-axes. The z-axis is the calculated net present value (NPV) for each engine in million US dollars. Figures 5.6-5.8 represent the different conditions of electric value with the modeled carbon taxes as subfigures. To indicate the

highest NPV, black contour lines have been overlaid onto each plot. However, not all conditions produce positive NPVs.

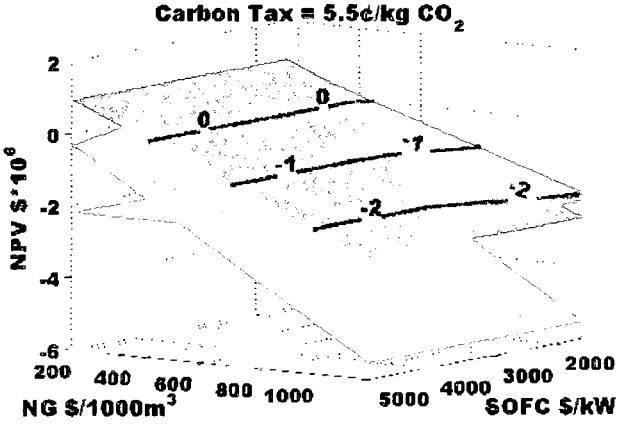
Initial studies found mostly negative NPVs for cases where electricity was sold for less than 15¢/kW·hr. Thus, where grid-supplied electricity is available, distributed generation is not economically competitive. Additionally, the simple microturbine never yields results as the highest NPV, due to its high cost and poor efficiency. However, this technology may still be viable when a large thermal heating load is needed, resulting in combined heat and power (CHP).

Through the improved efficiency of the ICR turbine and the extreme efficiency of the SOFCGT hybrid (even with the 10% penalty), these novel options present the highest NPV of the compared options when natural gas prices are below $937.28/1000m^3 ($27/MMBtu). This indicates that even with natural gas prices well above current market value, using the above technology may still be economically viable compared to diesel generation. As natural gas prices or the carbon tax increase, the higher efficiency of the SOFCGT hybrid makes it a better option. However, the NPV surfaces of these two engines are always generally close one to another.

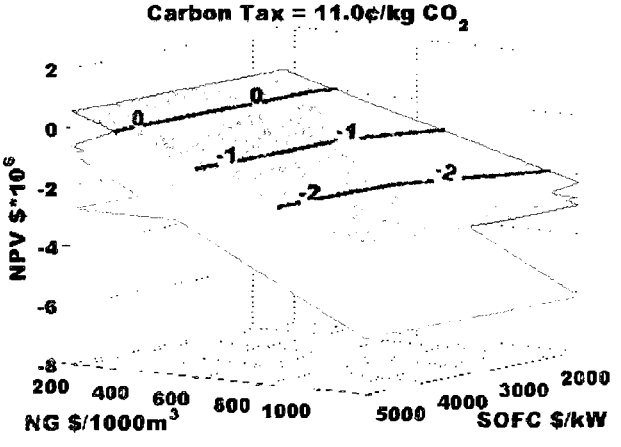
One can also conclude that diesel generators only appear as an attractive option when the value of electricity is above 25¢/kW·hr and the cost of natural gas exceeds $937.28/1000m^3.



5.6 a

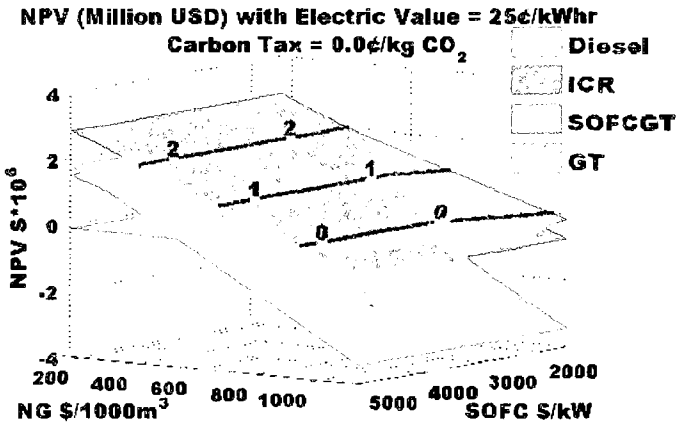


5.6 b

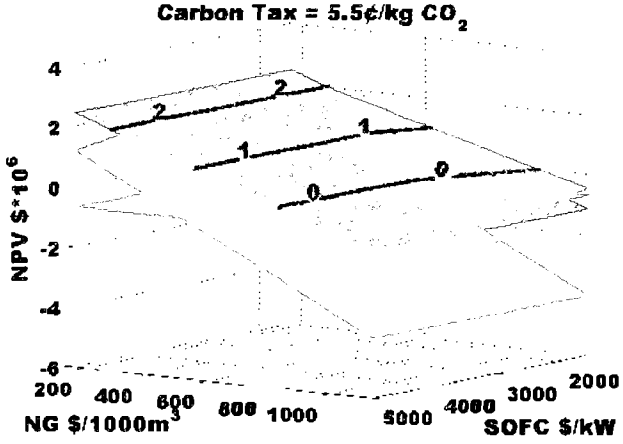


5.6 c

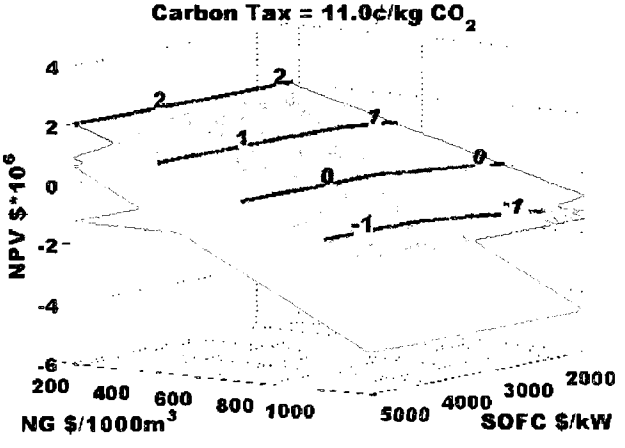
Figure 5.6: NPV with 15¢/kW hr electricity with varied carbon taxes



5.7 a

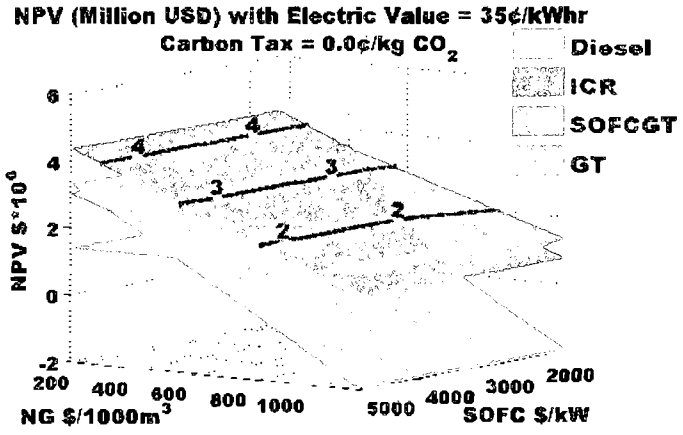


5.7 b

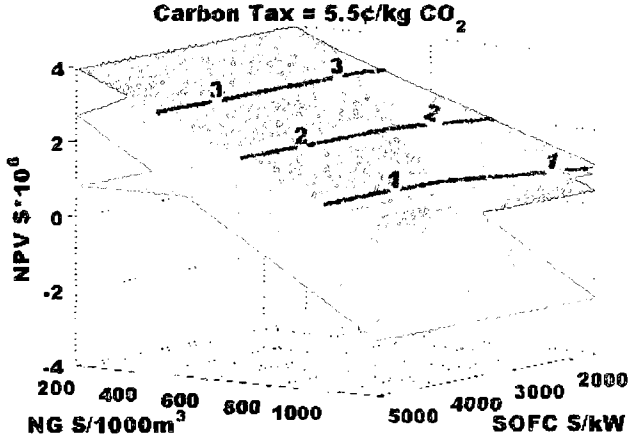


5.7 c

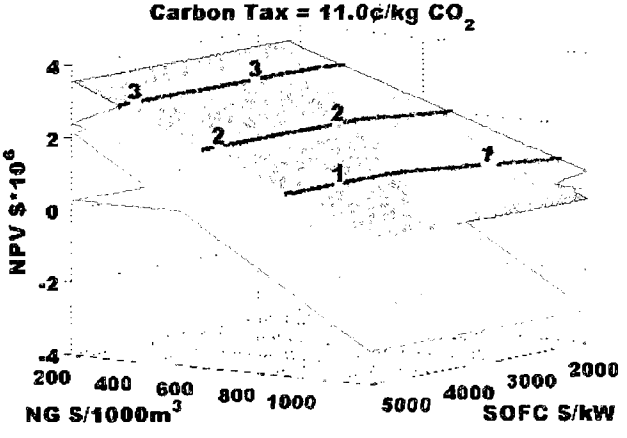
Figure 5.7: NPV with 25¢/kWhr electricity with varied carbon taxes



5.8 a



5.8 b



5.8 c

Figure 5.8: NPV with 35¢/kWhr electricity with varied carbon taxes

5.8. Discussion

The above results study NPV for distributed generation options. However, combined heat and power (CHP) applications were not investigated. One may argue that the inefficiencies of an engine could be used as heat for a village heating load. However, it is the author's experience that in many rural Alaskan villages recovered heat from the central power station is rarely used to supply more than a handful of community buildings, often including a school, health clinic, and Laundromat due to the high cost of moving heat. Additionally, as many Alaskan villages have very large heating loads, several times larger than their electric load, the sparse building density does not condone itself to district heating due to transportation distance losses and overall expense. Therefore, most villages rely on costly diesel fuel for residential heating which has a thermal efficiency approaching 85% LHV.

It is also important to consider that many gas turbines are capable of using alternative fuels such as diesel, biofuels, and less pure methane sources. The ICR 225, with higher efficiencies than diesel engines and predicted lower O&M costs, represents a potentially viable alternative to diesel engines.

The pairing of the ICR 225 to an SOFC, as discussed in reference [53], offers benefits of lower capital costs, high efficiency, a 5:1 turndown ratio, and excellent thermal management of the SOFC. Steady-state analysis in reference [53] shows that SOFC stack exit temperature can be maintained at 1273 K throughout a 5:1 turndown. While transient analysis is needed, the results offer hope that the system may respond quickly to transients, while avoiding additional equipment costs for flywheels or batteries.

It is the author's hope that when development of the ICR 225 Vehicle Engine advances to produce physical engines, a working ICR engine may be used to test performance and transients of the proposed SOFCGT hybrid configuration in a hardware-in-the-loop simulation, where the plenum volume and heat characteristics of the SOFC are simulated while connected to operational turbomachinery.

5.9. Conclusions

The results of this study clearly show the high cost of distributed generation. Figure 5.6 represent electrical rates of 15¢/kW·hr where NPVs are barely in positive territory when natural gas prices are low. With NPV so low for distributed generation, grid connected power remains the preferred option where available. In remote sites, if natural gas reserves are available, they may prove economical to develop, even when the price of this natural gas must be valued above market rates.

As one would expect, increased fuel costs and carbon taxes favor a higher efficiency engine, one that meets the load using less fuel while emitting less carbon. Across each plot of varied carbon tax and electrical value the ICR 225 Vehicle Engine usually represents the highest NPV. Only when carbon taxes are 11¢/kg CO_2 ($100/ton CO_2) and SOFC capital costs are below $1546/kW does the SOFCGT hybrid present a slightly higher NPV than the ICR 225.

The SOFCGT hybrid presents the highest NPVs when prices of natural gas are higher, CO_2 emissions are taxed more, and SOFC capital costs are low. The target price for mass manufacturing of SOFCs is $1400/kW, and when reliability and performance are guaranteed (the definition of a commercial product) the SOFCGT hybrid may find a commercial market.

In conclusion, the ICR 225 Vehicle Engine shows promise as an alternative to diesel generators and distributed generation engines. As a carbon tax is introduced, higher efficiency engines will prove more economical. Due to the uncertainty in the fuel cell industry, it is unclear when demonstrations will resume or products will become marketable. However, when another attempt is made to design a pressurized SOFC gas turbine hybrid, the proposed SOFCGT hybrid configuration deserves consideration.

5.10. Appendix: Additional Figures

When submitting the above journal article, reviewer comments suggested to use three-dimensional plots of the NPV results, as is shown in Figures 5.6-5.8. However, the author believes the results can be better interpolated from a top-down, two-dimensional, view as shown in Figures 5.9-5.17 below.

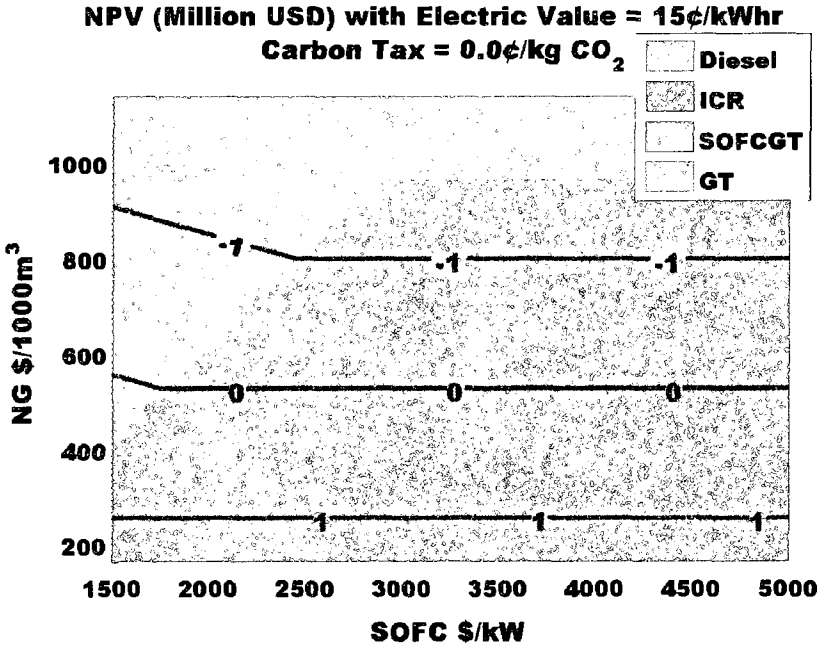


Figure 5.9: NPV with 15¢/kWhr & 0.0¢/kg CO_2

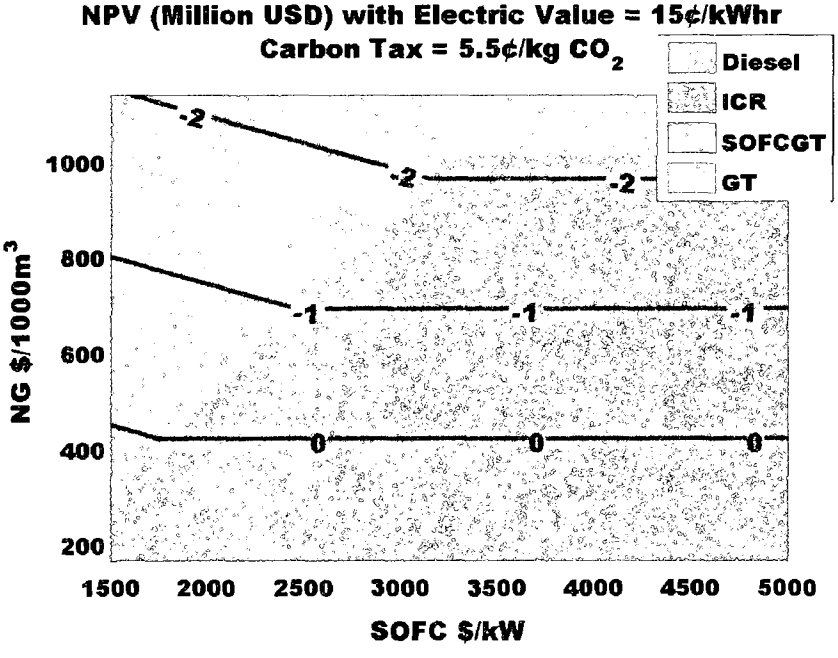


Figure 5.10: NPV with 15¢/kWhr & 5.5¢/kg CO_2

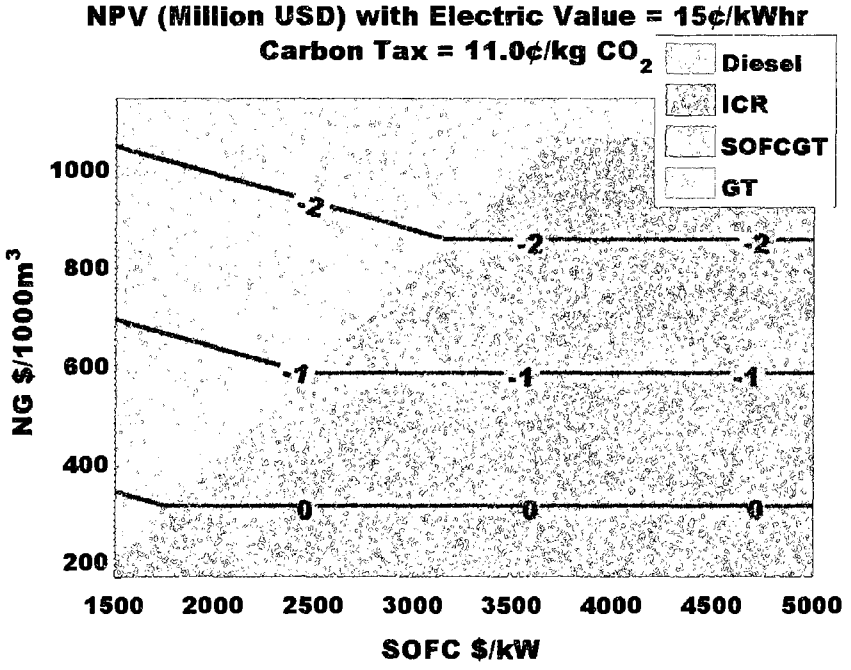


Figure 5.11: NPV with 15¢/kWhr & 11.0¢/kg CO_2

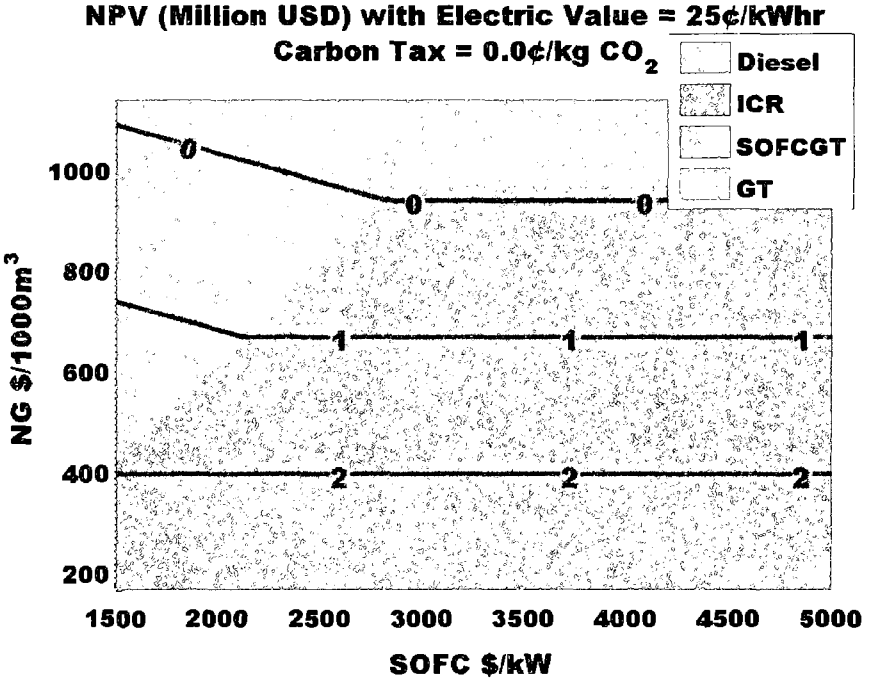


Figure 5.12: NPV with 25¢/kWhr & 0.0¢/kg CO_2

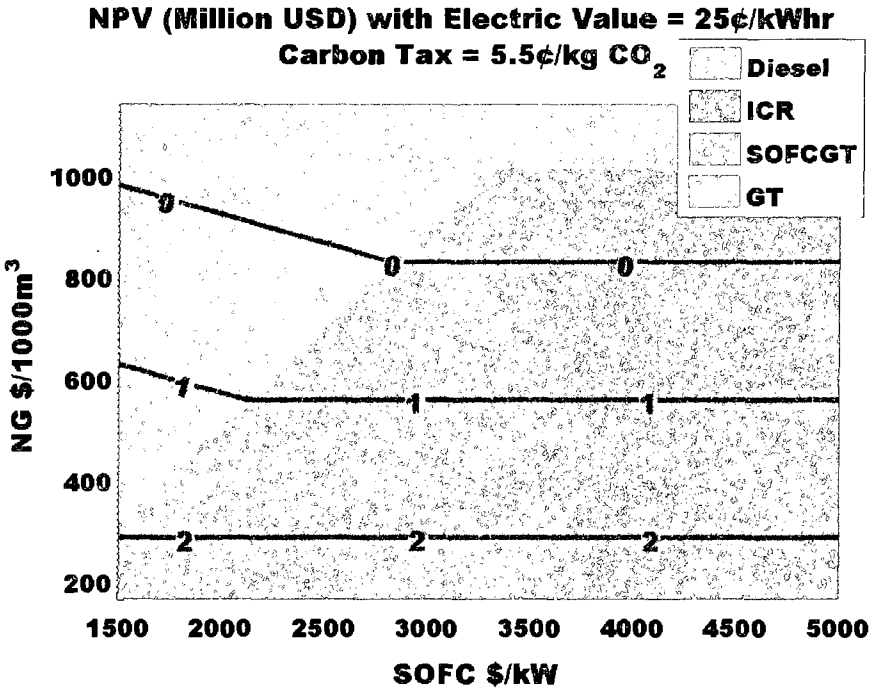


Figure 5.13: NPV with 25¢/kWhr & 5.5¢/kg CO_2

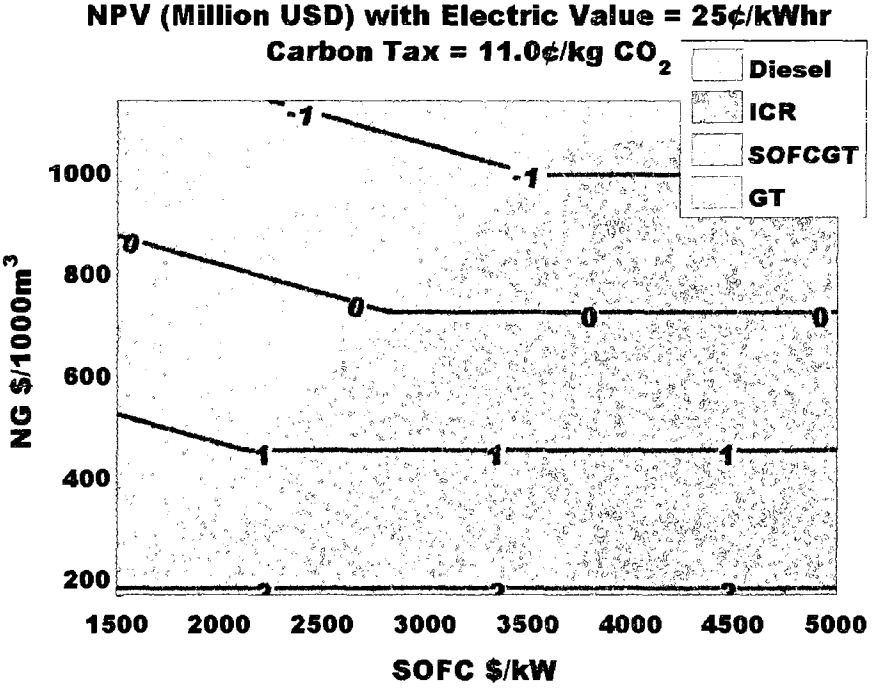


Figure 5.14: NPV with 25¢/kWhr & 11.0¢/kg CO_2

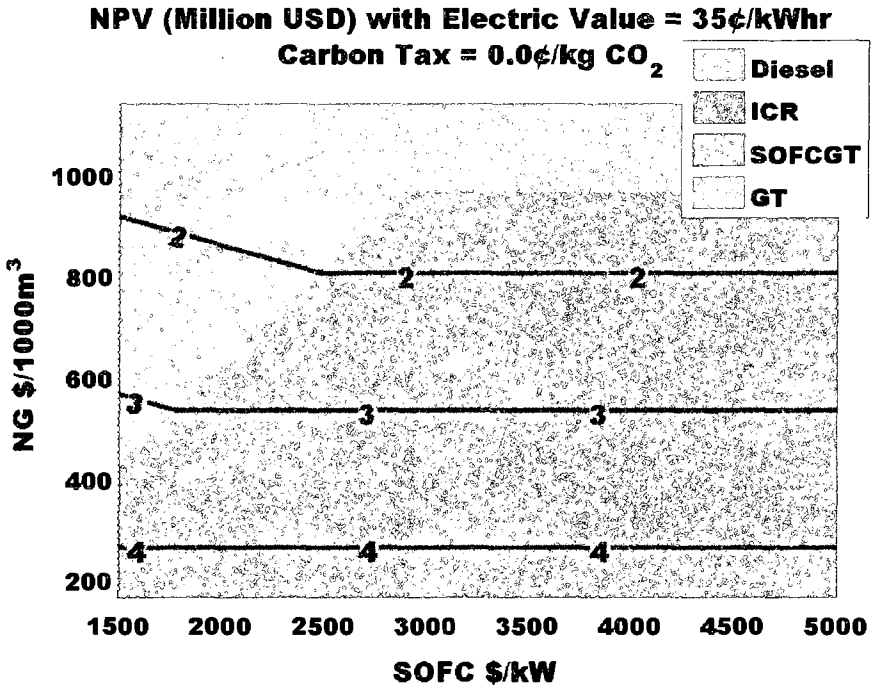


Figure 5.15: NPV with 35¢/kWhr & 0.0¢/kg CO_2

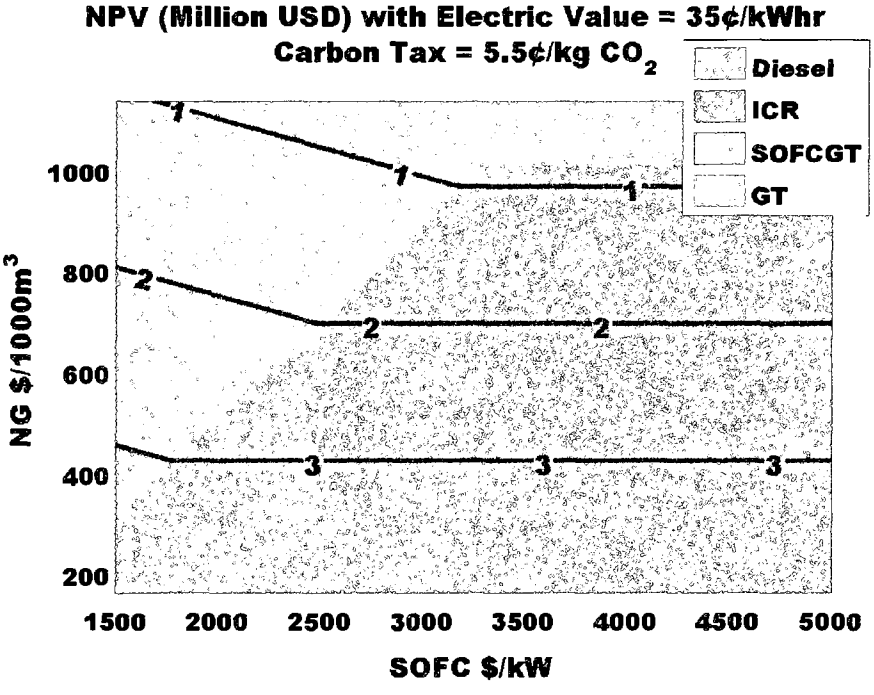


Figure 5.16: NPV with 35¢/kWhr & 5.5¢/kg CO_2

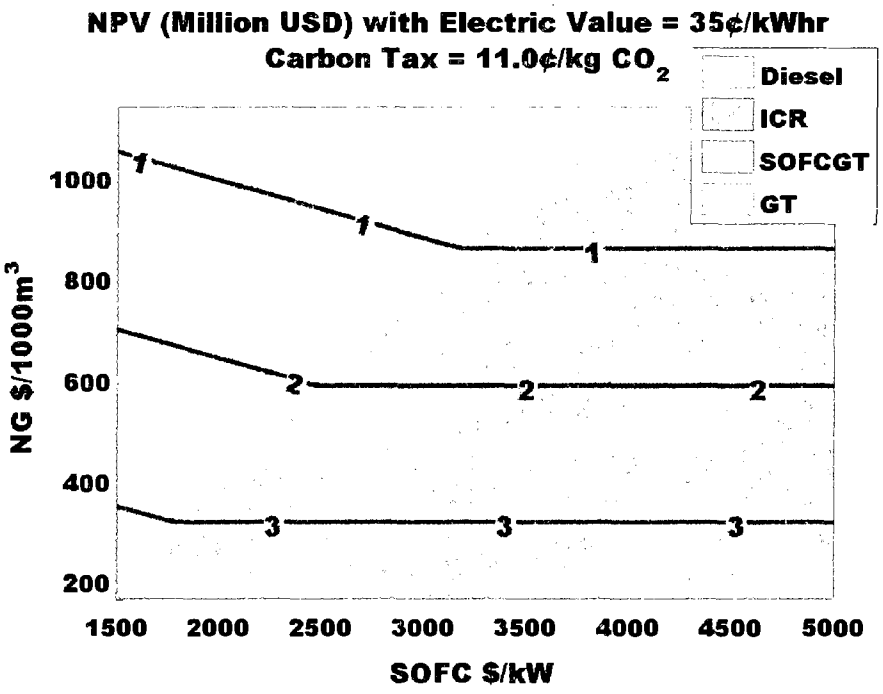


Figure 5.17: NPV with 35¢/kWhr & 11.0¢/kg CO_2

CHAPTER 6: Final Conclusions

This thesis has investigated a novel solid oxide fuel cell gas turbine (SOFC-GT) hybrid. Performance results presented in Chapter 4 show a remarkable electrical turndown ratio of 5:1, while the SOFC cathode exit temperature is maintained at 1000°C and the temperature differential across the SOFC stack is retained at less than 200°C. This represents a new achievement in SOFC-GT hybrid configurations. Previously proposed systems have reduced fuel cell cathode exit temperatures in order to achieve even small turndowns, and have often used bleed and/or by-pass valves to correct the discrepancy between turbomachinery airflow and the cooling needs of the SOFC.

The proposed SOFC-GT hybrid addresses several issues which surfaced in the demonstration of the Siemens pressurized-hybrid (PH220) at the University of California, Irvine. The most notable improvement is the matching of the airflow provided by the turbomachinery to the cooling requirements of the SOFC. This is achieved through the use of two control variables: first, the auxiliary combustor to regulate the thermal and power demands of the turbomachinery separately from that of the SOFC, and secondly, the variable geometry turbine to regulate the airflow of the turbomachinery to match the cooling needs of the SOFC. Because the SOFC cathode exit temperature is maintained throughout turndown, thermal transients within the SOFC are expected to be minimal.

At low power generation levels the SOFC-GT hybrid resembles previously proposed hybrids, in that the turbomachinery contributes only 20% of the generated power, with the SOFC contributing 80%. However, the newly proposed SOFC-GT hybrid is unique in that, at design point, the inexpensive turbomachinery contributes slightly more than half of the generated power. This is expected to significantly reduce capital costs over previous systems. Thus, the capital cost per kilowatt of the SOFC-GT hybrid is the cost per kilowatt average between the ICR turbomachinery and the SOFC stack costs.

Chapter 5 gives a coarse economic analysis comparing the SOFC-GT hybrid to several other engines, including the ICR turbine engine, a simple microturbine, and a diesel engine. Results show that the improved efficiencies and lower costs of the ICR

engine (over that of a diesel engine) have allowed the ICR to present higher net present values (NPV) even when natural gas costs are significantly higher than Henry Hub market rates. This suggests that rural areas should develop a natural gas resource to partially displace use of diesel fuel, even if prices of the natural gas would be well above national rates.

Further analysis should be made into several assumptions made in the UAF model, including the constant heat exchanger effectiveness, the constant shaft-bearing losses, and the constant intercooler-blower parasitic load. The UAF model currently assumes that these values remain fixed throughout the study. However, future improvements to the model could allow these values to fluctuate with the off-design analysis.

Three suggestions made by colleagues for further research are discussed below:

- What would change in the operation of the current SOFC-GT configuration with the free-pressure turbine constrained to rotate at a fixed shaft speed? This would allow the connected generator to produce synchronized AC electricity. The desirability of this is the difficulty of converting variable AC power into useable AC electricity. However, the fuel cell will always require electrical conversion from DC into AC power. The current assumption is that the variable speed generator power will be converted into DC and added to the DC power from the SOFC. Both sources of DC power will then be converted into usable AC.
- How would the system behave if the prescribed SOFC cathode exit temperature were lowered? The development of new SOFC electrolyte materials aims to lower operating temperatures of the SOFC stack. This decreases the thermal expansion stress between the electrolyte and interconnect materials. Preliminary results show similar engine performance, with a slight decrease in efficiency when the prescribed cathode exit temperature is lowered.
- How would the system perform if the variable turbine inlet nozzles were used on the low-pressure turbine, and the free-pressure turbine were removed from the system? This would decrease the number of components and remove one of the inefficiencies from the system. Consequences of this action are largely unknown. However, the ceramic rotor used within the high-pressure turbine is more fragile than its metal-alloy

counterpart; thus, the pressure differential (expansion ratio) across the ceramic rotor must be limited. It is likely that removing the free-pressure turbine would increase the expansion ratio across both the high- and low-pressure turbine and possibly exceed the limits of the ceramic rotor.

The SOFC-GT hybrid is a unique configuration that deserves further consideration. Brayton Energy LLC hopes to build a prototype of the ICR engine when funding becomes available. It is the author's hope that an ICR prototype will be coupled with a hardware-in-the-loop simulated SOFC, in order to study the transient dynamics of the proposed SOFC-GT hybrid. The author feels that operating the physical turbomachinery with a simulated SOFC would yield results significantly more realistic than using a transient model alone.

LITERATURE CITED

1. Siemens. *Siemens Power Generation - SOFC Systems Manufacturing.* 2009; Available from: http://www.powergeneration.siemens.com/products-solutions-services/products-packages/fuel-cells/demonstrations/#25kWSOFCsystematNationalFuelCellResearchCenter(NFCRC),Irvine,California.

2. Litzinger, K.P. *Commercially Emerging Hybrid Systems.* in *ASME ICEPAG.* 2005. Irvine, CA.

3. Siemens. *Siemens Power Generation - SOFC Solutions Provider.* 2008 [cited 2008 5/29/08]; Available from: http://www.powergeneration.siemens.com/products-solutions-services/products-packages/fuel-cells/demonstrations/demonstrations-summary/.

4. George, R. and A. Casanova, *Developments in Siemens Westinghouse SOFC Program,* in *2003 Fuel Cell Seminar,* S. Westinghouse, Editor. 2003.

5. Litzinger, K.P. *Turbo Machinery Hybrid Component Technology.* in *ICEPAG.* 2005. Irvine, CA: ASME.

6. Agnew, D.G.D. *Novel Components for Reduced Costs.* in *ICEPAG.* 2004. Irvine, CA.

7. Berenyi, S.G. *Progress on a High Pressure Turbocharger for SOFC/ Hybrid Fuel Cell Systems.* in *ICEPAG.* 2004. Irvine, CA.

8. Agnew, D.G.D. and M. Bozzolo. *Design and Integration -Scaling of Hybrid Systems.* in *ICEPAG.* 2004. Irvine, CA.

9. Ghezel-Ayagh, H., *Prospects for Ultra High Efficiency Power Generation,* F. Energy, Editor: Danbury, CT.

10. Ghezel-Ayagh, H., *Project Fact Sheet Direct FuelCell/Turbine® Power Plant,* F. Energy, Editor. 2006, FuelCell Energy: Danbury.

11. Ghezel-Ayagh, H. and S.T. Junker, *IV.E.2 Advanced Control Modules for Hybrid Fuel Cell/Gas Turbine Power Plants,* O.o.F.E.F.C. Program, Editor. 2007.

12. Mueller, F., et al., *Control Design for a Bottoming Solid Oxide Fuel Cell Gas Turbine Hybrid System.* Journal of Fuel Cell Science and Technology, 2007. **4**(August 2007): p. 10.

13. Roberts, R., et al., *Control design of an atmospheric solid oxide fuel cell/gas turbine hybrid system: Variable versus fixed speed gas turbine operation.* Journal of Power Sources, 2006. **161**(1): p. 484-491.

14. Mueller, F., et al. *Linear Quadratic Regulator for a Bottoming Solid Oxide Fuel Cell Gas Turbine Hybrid System.* in *7th International Colloquim on Environmentally Preffered Advanced Power Generation (ICEPAG).* 2006. Newport Beach, CA: ASME.

15. Liang, A.D. *NASA Aircraft Fuel Cell Development.* in *ICEPAG.* 2004. Irvine, CA.

16. Daggett, D. *Fuel Cell APU (FCAPU) Overview & Plan.* in *ICEPAG.* 2004. Irvine, CA.

17. Au, S.F., et al., *The influence of operating temperature on the efficiency of a combined heat and power fuel cell plant.* Journal of Power Sources, 2003. **122**(1): p. 37-46.

18. Bessette II, N.F., *Modeling and Simulation for Solid Oxide Fuel Cell Power Systems*, in *Mechanical Engineering.* 1994, Georgia Institute of Technology. p. 233.

19. Costamagna, P., L. Magistri, and A.F. Massardo, *Design and part-load performance of a hybrid system based on a solid oxide fuel cell reactor and a micro gas turbine.* Journal of Power Sources, 2001. **96**(2): p. 352-368.

20. Grillo, O., L. Magistri, and A.F. Massardo, *Hybrid systems for distributed power generation based on pressurisation and heat recovering of an existing 100 kW molten carbonate fuel cell.* Journal of Power Sources, 2003. **115 (2003)**(252-267).

21. Magistri, L., et al., *Design and Off-Design Analysis of a MW Hybrid System Based on Rolls-Royce Integrated Planar Solid Oxide Fuel Cells.* Journal of Engineering for Gas Turbines and Power, 2007. **129**(July): p. 6.

22. Wolf, T., *Cycle/Performance Analysis*, B.E. Systems, Editor. 2005.

23. Wolf, T., et al., *Design Study for an Intercooled and Recuperated Microturbine as a Stand-Alone Engine and Incorporated into a Hybrid Solid-Oxide Fuel Cell Powerplant.* 2007, Arctic Energy Technology Development Laboratory University of Alaska Fairbanks. p. 58.

24. Wolf, T.L., J.B. Kesseli, and J.S. Nash. *Preliminary Deign and Projected Performance for Intercooled-Recuperated Microturbine.* in *ASME TurboExpo 2008.* 2008. Berlin, Germany: ASME GT2008-50527.

25. Lin, P.-H. and C.-W. Hong, *On the start-up transient simulation of a turbo fuel cell system.* Journal of Power Sources, 2006. **160**(2): p. 1230-1241.

26. Tucker, D., et al., *Evaluation of Hybrid Fuel Cell Turbine System Startup with Compressor Bleed*, in *ASME Turbo Expo*. 2005, ASME: Reno, NV.

27. Roberts, R., et al., *Dynamic Simulation of Carbonate Fuel Cell-Gas Turbine Hybrid Systems.* Journal of Engineering for Gas Turbines and Power, 2006. **128**(April 2006): p. 8.

28. Roberts, R.A., et al., *Development of Controls for Dynamic Operation of Carbonate Fuel Cell-Gas Turbine Hybrid Systems*, in *ASME Turbo Expo*. 2005, ASME: Reno, NV.

29. Traverso, A., et al., *Gas Turbine Assessment for Air Management of Pressurized SOFC/GT Hybrid Systems.* Journal of Fuel Cell Science and Technology, 2007. **4**(November 2007): p. 11.

30. Mueller, F., et al., *Dynamic Simulation of an Intergrated Solid Oxide Fuel Cell System Including Current-Based Fuel Flow Control.* Journal of Fuel Cell Science and Technology, 2006. **3**(May 2006): p. 11.

31. Ferrari, M.L., et al., *Transient Modeling of the NETL Hybrid Fuel Cell/Gas Turbine Facility and Experimental Validation.* Journal of Engineering for Gas Turbines and Power, 2007. **129**(October 2007): p. 8.

32. Liese, E., *Tour of National Energy Technology Laboratory for EPSCor*, W. Burbank, Editor. 2005: Morgantown, WV.

33. Shelton, M., et al., *A Transient Model of a Hybrid Fuel Cell/Gas Turbine Test Facility Using Simulink*, in *ASME Turbo Expo*. 2005, ASME: Reno, NV.

34. Gemmen, R.S., et al. *Development of Dynamic Modeling Tools for Solid Oxide and Molten Carbonate Hybrid Fuel Cell Gas Turbine Systems.* in *International Gas Turbine & Aeroengine Congress & Exhibition*. 2000. Munich, Germany.

35. Trembly, J.P., R.S. Gemmen, and D.J. Bayless, *The effect of IGFC warm gas cleanup system conditions on the gas–solid partitioning and form of trace species in coal syngas and their interactions with SOFC anodes.* Journal of Power Sources, 2007. **163**: p. 11.

36. Calise, F., et al., *Single-level optimization of a hybrid SOFC-GT power plant.* Journal of Power Sources, 2006. **159**(2): p. 1169-1185.

37. Massardo, A.F., C.F. McDonald, and T. Korakianitis, *Microturbine/Fuel-Cell Coupling for High-Efficiency Electrical-Power Generation.* Journal of Engineering for Gas Turbines and Power, 2002. **124**(January): p. 7.

38. Marsano, F., L. Magistri, and A.F. Massardo, *Ejector performance influence on a solid oxide fuel cell anodic recirculation system.* Journal of Power Sources, 2004. **129**(2004): p. 216-228.

39. Traverso, A., A.F. Massardo, and R. Scarpelli, *Externally Fired Micro-Gas Turbine: Modelling and Experimental Performance.* Applied Thermal Engineering, 2006. **26**(2006): p. 1935-1941.

40. Traverso, A., R. Scarpellini, and A. Massardo, *Experimental Results and Transient Model Validation of an Externally Fired Micro Gas Turbine*, in *AMSE Turbo Expo 2005*. 2005, ASME: Reno, NV.

41. Pascenti, M., et al. *MICRO GAS TURBINE BASED TEST RIG FOR HYBRID SYSTEM EMULATION.* in *ASME Turbo Expo 2007*. 2007. Montreal Canada.

42. Ferrari, M.L., et al., *Control System for Solid Oxide Fuel Cell Hybrid Systems.* ASME Conference Proceedings, 2005. **2005**(47284): p. 55-63.

43. Magistri, L., F. Trasino, and P. Costamagna, *Transient Analysis of Solid Oxide Fuel Cell Hybrids—Part I: Fuel Cell Models.* Journal of Engineering for Gas Turbines and Power, 2006. **128**(April 2006): p. 5.

44. Magistri, L., et al., *Heat Exchangers for Fuel Cell and Hybrid System Applications.* Journal of Fuel Cell Science and Technology, 2006. **3**(May): p. 8.

45. Traverso, A., *TRANSEO Code for the Dynamic Performance Simulation of Micro Gas Turbine Cycles*, in *ASME Turbo Expo*. 2005, ASME: Reno, NV.

46. Bozzolo, M. *System and Components Modeling.* in *ICEPAG*. 2004. Irvine, CA.

47. Hirschenhofer, J.H., et al., *Fuel Cell Handbook Fourth Edition*, U.S.D.o. Energy, O.o.F. Energy, and F.E.T. Center, Editors. 1998, Parsons Corporation.

48. Burbank, W., *Building a Toolset for Fuel Cell Turbine Hybrid Modeling*, in *Mechanical Engineering*. 2006, University of Alaska Fairbanks: Fairbanks. p. 156.

49. Morgan, M.J. and H.N. Shapiro, *Fundamentals of Engineering Thermodynamics*. 2000: John Wiley & Sons.

50. Smith, J.M., H.C.V. Ness, and M. Abbott, *Introduction to Chemical Engineering Thermodynamics*. 2000: McGraw-Hill Science/Engineering/Math.

51. Wolf, T., *Ramgen Off-Design Modeling*, B.E. Systems, Editor. 2006.

52. EG&G Technical Services, I., *Fuel Cell Handbook (Seventh Edition)*, DOE, Editor. 2004: Morgantown, West Virginia.

53. Burbank, W.S., D.E. Witmer, and F. Holcomb, *Model of a Novel Pressurized SOFC-GT Hybrid Engine.* Journal of Power Sources, 2009. **POWER-D-09-00248R1**.

54. Larminie, J. and A. Dicks, *Fuel Cell Systems Explained.* 2003: John Wiley & Sons Ltd.

55. Hsu, M.S., *Pressurized, Integrated Electrochemical Converter Energy System*, U.S.P. Office, Editor. 1999, Ztek Corporation, Waltham, Mass. p. 16.

56. Hsu, M.S. and E.D. Hoag, *Ultra-High Efficiency Turbine and Fuel Cell Combination*, U.S.P. Office, Editor. 1997, Ztek Corporation, Waltham, Mass. p. 14.

57. Hsu, M.S. and E.D. Hoag, *Electrochemical Converter Having Internal Thermal Integration*, U.S.P. Office, Editor. 1996, Ztek Corporation, Waltham, Mass. p. 12.

58. Ferrari, M.L., et al., *Influence of the anodic recirculation transient behaviour on the SOFC hybrid system performance.* Journal of Power Sources, 2005. **149**(2005): p. 22-32.

59. Kaneko, T., J. Brouwer, and G.S. Samuelsen, *Power and temperature control of fluctuating biomass gas fueled solid oxide fuel cell and micro gas turbine hybrid system.* Journal of Power Sources, 2006. **7773**.

60. Calise, F., et al., *Full load synthesis/design optimization of a hybrid SOFC-GT power plant.* Energy, 2007. **32**(4): p. 446-458.

61. Christopher J. Steffen, J., J.E. Freeh, and L.M. Larosiliere, *Solid Oxide Fuel Cell/Gas Turbine Hybrid Cycle Technology for Auxiliary Aerospace Power*, in *Turbo Expo 2005*. 2005, NASA: Reno, NV.

62. Kim, T.S. and S.H. Hwang, *Part load performance analysis of recuperated gas turbines considering engine configuration and operation strategy.* Energy, 2006. **31**(2-3): p. 260-277.

63. Veyo, S.E., et al., *Tubular Solid Oxide Fuel Cell/Gas Turbine Hybrid Cycle Power Systems: Status.* Journal of Engineering for Gas Turbines and Power, 2002. **124**(October): p. 5.

64. Yang, J.S., J.L. Sohn, and S.T. Ro, *Performance characteristics of a solid oxide fuel cell/gas turbine hybrid system with various part-load control modes.* Journal of Power Sources, 2007. **166**(1): p. 155-164.

65. Yang, J.S., J.L. Sohn, and S.T. Ro, *Performance characteristics of part-load operations of a solid oxide fuel cell/gas turbine hybrid system using air-bypass valves.* Journal of Power Sources, 2008. **175**(1): p. 296-302.

66. Zhang, X., et al., *Dynamic modeling of a hybrid system of the solid oxide fuel cell and recuperative gas turbine.* Journal of Power Sources, 2006. **163**(1): p. 523-531.

67. Williams, M.C., J. Strakey, and W. Sudoval, *U.S. DOE fossil energy fuel cells program.* Journal of Power Sources, 2006. **159**(2): p. 1241-1247.

68. Janes, J., *A Fully Enhanced Gas Turbine For Surface Ships*, in *ASME International Gas Turbine And Aeroengine Congress & Exhibition.* 1996: Birmingham, U.K.

69. Navy, C.C.R.E.R., *The WR-21 Intercooled Recuperated Gas Turbine Engine – Integration Into Future Warships.* Proceedings of the International Gas Turbine Congress 2003 Tokyo, 2003.

70. Rolls-Royce, *WR-21 Fact Sheet.* 2000.

71. Goodwin, D.G., *Cantera C++ User's Guide.* 2002.

72. Kofsky, M.G. and W.J. Nusbaum, *Aerodynamic Evaluation of Two-Stage Axial-Flow Turbine Designed for Brayton-Cycle Space Power System*, NASA, Editor. 1968.

73. Rohlik, H.E., *Radial-Inflow Turbines*, NASA, Editor. 1972. p. 31-57.

74. Szanca, E.M. and H.J. Schum, *Experimental Determination of Aerodynamic Performance*, NASA, Editor. 1972. p. 103-139.

75. Meitner, P.L. and A.J. Glassman, *Off-Design Performance Loss Model for Radial Turbines with Pivoting, Variable-Area Stators*, N.L.R. Center, Editor. 1980: Cleveland.

76. Singhal, S.C. and K. Kendall, *High-Temperature Solid Oxide Fuel Cells: Fundamentals, Design and Applications*. 2003: Elsevier.

77. MicroTurbine, C., *Capstone MicroTurbine Model 330 Installation & Start-Up.* 1990, Capstone MicroTurbine: Chatsworth, CA. p. 53 - 59.

78. Gillette, S., *ICR 225 Vehicle Engine FactSheet.* 2008, Capstone MicroTurbines.

79. Devine, M. and E.I. Baring-Gould, *The Alaska Village Electric Load Calculator*, N.R.E. Laboratory, Editor. 2004: Golden, Colorado.

80. Palsson, J., *Thermodynamic Modelling and Performance of Combined Soid Oxide Fuel Cells and Gas Turbine Systems*, in *Department of Heat and Power Engineering*. 2002, Lund University: Lund, Sweden. p. 67.

81. Singhal, S.C., *Advances in solid oxide fuel cell technology.* Solid State Ionics, 2000. **135**(1-4): p. 305-313.

82. Singhal, S.C., *Solid oxide fuel cells for stationary, mobile, and military applications.* Solid State Ionics, 2002. **152-153**: p. 405-410.

83. Stevenson, J.W., P. Singh, and S.C. Singhal, *The secrets of SOFC success*, in *The Fuel Cell Review*. 2005.

84. DoD. *DoD Fuel Cell - Residential Demonstration Projects.* 2006; Available from: http://dodfuelcell.cecer.army.mil/res/index.php4.

85. Appleby, A.J. and K. Foger, *A Fuel Cell Handbook 2nd edition*, ed. K.P. Co. 1993, New York.

86. Agrawal, G., et al., *IV.E.4 Foil-Bearing Supported High-Speed Centrifugal Cathode Air Blower*, O.o.F.E.F.C. Program, Editor. 2007.

87. Agrawal, G., et al., *IV.E.5 Foil Gas Bearing Supported High-Speed Centrifugal Anode Gas Recycle Blower*, O.o.F.E.F.C. Program, Editor. 2007.

88. Alinger, M., *III.5 Solid State Energy Conversion Alliance (SECA) Solid Oxide Fuel Cell Program*, O.o.F.E.F.C. Program, Editor. 2007.

89. Alinger, M. and J. Powers, *II.2 Solid Oxide Fuel Cell Coal-Based Systems*, O.o.F.E.F.C. Program, Editor. 2007.

90. Allen R. Hefner, J., *IV.C.2 Advanced Power Conversion System (PCS) Technologies for High-Megawatt Fuel Cell Power Plants*, O.o.F.E.F.C. Program, Editor. 2007.

91. Alptekin, G., *IV.B.7 Sorbents for Desulfurization of Natural Gas and LPG*, O.o.F.E.F.C. Program, Editor. 2007.

92. Brouwer, J. and R. Roberts, *Personal Communication: Matlab and Simulink Tutorial*, W. Burbank, Editor. 2004, National Fuel Cell Research Center: Irvine, CA.

93. NREL, *Getting Started Guide for Homer Version 2.1*. 2005, National Renewable Energy Laboratory. p. 30.

94. Leasy, E., *Tour of National Energy Technology Laboratory for EPSCor*, W. Burbank, Editor. 2005: Morgantown, WV.

APPENDIX A: Fuel Cell Technology Overview

A.1. History of Fuel Cells

Sir William Grove invented fuel cells in 1838, and experiments in 1906 focused on the direct conversion of coal to electricity. However, the first successful technical application of fuel cells was onboard the NASA Apollo spaceships during the 1960's [80]. Today, fuel cells are being researched and developed in hopes of both reducing pollutants and increasing electrical efficiency, thereby using less fuel. Low temperature fuel cells such as proton exchange membrane fuel cells (PEMFC) are currently being demonstrated in city busses and a limited number of personal vehicles. However, in the spring of 2009, the Obama administration canceled all fuel cell funding toward automotive use citing the technology as not being ready for market in five to ten years. High temperature fuel cells are most practical in stationary applications where the major hurdles to market acceptance remain cost and reliability (hurdles which have not changed in over twenty years).

A.2. Fuel Cell Overview

Fuel cells are devices that convert the chemical energy within fuels into electrical and heat energies. The conversion efficiency of chemical energy to electrical energy is theoretically much higher using fuel cells than with other alternatives, such as internal combustion engines or steam power plants. However, it is important to remember that no process is perfectly efficient, and that losses in the form of heat are always a byproduct. While the limits on heat engines are given by the second law of thermodynamics and the Carnot cycle, the limits on electrochemical efficiency are described by the Gibbs Free energy limits, the third law of thermodynamics.

Fuel cells have unfortunately been termed "hydrogen batteries", so the public often imagines a sealed box similar to a car battery. However, fuel cells can only produce electricity when both a fuel and an oxygen source (air) are provided, a system which more closely parallels an automotive engine with a gasoline tank. Fuel cell systems exhaust the spent reactants (air and fuel), showing further similarities to an automotive

engine. Fuel cells have also been improperly advertised as systems with no moving parts. However, compressors, fans, or blowers are used to supply both air and fuel to the fuel cell in precise flow rates. These devices move, wear out, and make noise. Thus, while the actual fuel cell may be solid state, the components required for operation of a fuel cell do move. All in all, fuel cells more closely resemble an engine than a battery.

However, fuel cells do in fact resemble a battery in the direct production of electricity (direct current, DC). And the fuel cell can be operated in reverse, consuming electricity to produce fuel and oxygen, thus recharging a fuel tank. (However, recharging efficiencies are considerably worse than consumer batteries, and hydrogen compression and storage is difficult).

Fuel cells have received much attention due to their lower pollution and higher electrical efficiencies. However, these benefits coexist with several negative consequences, most noticeably in its difference from combustion. A fuel cell carries out chemical reactions similar as those that take place in combustion. However, the electrochemical reactions of a fuel cell require heterogeneous chemical reactions to occur on a surface, either the anode or the cathode. The fuel cell's reacting surface is susceptible to becoming contaminated by impurities within the fuel and air supplies, slowing reaction rates and decreasing efficiency. Eventually, the surface may become so damaged that the fuel cell must be replaced. Comparatively, an internal combustion engine is a three dimensional homogenous reaction, where molecules of fuel and air randomly meet in space and explosively transfer electrons, releasing heat. Any impurities simply become part of the exhaust with all the other combustion products. This consequence of fuel cell function requires both fuel and air streams to be sufficiently pure to prevent contamination, and usually must undergo thorough cleaning/scrubbing to remove impurities, which both increases system costs and lowers efficiencies.

In demonstrations, fuel cells have shown a slight improvement in efficiency over internal combustion engines (about 46% for an SOFC operating on natural gas, vs. approximately 40% for a modern efficient diesel engine). However, roughly 50% of the fuel energy consumed in a fuel cell is converted to heat. This heat must be carefully

managed to prevent overheating of the fuel cell. Some low- and medium-temperature fuel cells use liquid cooling channels for heat removal; most high-temperature fuel cells rely on excess airflow to remove heat through the exhaust. The excess air travels into and through the fuel cell, absorbing heat from within the fuel cell before exhausting it into the atmosphere. In other fuel cells, temperature management is crucial as the electrolyte may be unable to conduct at cooler temperatures, and the interconnect components may begin to fail at higher temperatures. Airflow must be carefully controlled to remove heat while maintaining the high-temperature fuel cells at their operational temperatures.

A.2.1. Basic Fuel Cell Technology

All fuel cells employ the same basic design, consisting of three important layers: the anode, electrolyte, and cathode. Sandwiching these layers together forms the fuel cell stack. The anode is positively charged and is where the fuel interacts. The anode assists reactions while conducting both ions and electrons. The electrolyte is the most important layer, as it must allow conduction of ions but not electrons. This allows the electrons to travel through a conductor, supplying electricity to an electrical circuit. The cathode is negatively charged and is in contact with the oxygen source. The cathode also must conduct both electrons and ions while assisting in chemical reactions. Often a separate catalyst, such as platinum, is added to the anode and/or cathode to assist in increasing the speed of the chemical reactions, especially for lower temperature fuel cells.

The electrolyte layer is what gives the name to and differentiates the different fuel cell technologies. Thus a molten carbonate fuel cell uses a molten carbonate as the electrolyte. However, the operating temperature defines the applications for which a fuel cell may be best suited toward.

A.2.2. Fueling Fuel Cells

Most fuel cells use hydrogen gas. However, pure hydrogen is not readily found in nature, is expensive to compress and/or liquefy, and must be carefully stored, as slow leaks indoors can lead to explosive mixtures. Moreover, before considering storage and

transportation, even the generation of hydrogen is a difficult and expensive process. Electrolyzing water to produce hydrogen is commonly thought to be easy (splitting water into hydrogen and oxygen). However, proponents often neglect the large amount of electricity required to perform the electrolysis, where the net efficiency of electrolyzing hydrogen and using the hydrogen in a fuel cell to produce electricity is at best 50%. Hydrogen may also be produced from hydrocarbons through a reformation process from fuels such as methane (natural gas), propane, gasoline and diesel, and even coal. However, efficiency losses associated with reformation occur, and CO_2 is a byproduct of the reformation processes. Without CO_2 capture and sequestration, the greenhouse gas benefits of operating a fuel cell are negated. When reporting results, fuel cell supporters often neglect both the efficiency losses and the additional CO_2 emitted in the process of acquiring pure hydrogen fuel.

Interest in high temperature fuel cells is '*fueled*' by the ability for fuels such as methane, propane, and several other light hydrocarbons to be internally reformed. The high temperatures, limited oxygen, and catalytic surfaces aid in the reformation; the internal reformation decreases both the cost of additional equipment and the external efficiency losses.

A.3. Low-Temperature Fuel Cell Technology

Low-temperature fuel cells (between 20°C and 120°C) are quick both to turn on and to respond to changing electrical loads. While theoretical efficiencies are roughly 70%, parasitic loads reduce the overall system efficiency to about 50%, and when the hydrogen production is considered, the efficiency drops to about 25%, well below that of a standard internal combustion engine. Low-temperature fuel cells are best suited for mobile and on-demand power situations. Proton exchange membranes (PEM) technology is the most popular technology in the low temperature region. However, direct methanol fuel cells (DMFC) also operate in the low-temperature region

A.3.1. Proton Exchange Membrane Technology

Proton exchange membrane fuel cells (PEMFCs) are nearly all constructed using bipolar plate geometry [52]. Most PEMFCs utilize a perfluorosulficonic acid polymer as the electrolyte, most commonly Nafion® as sold by Dupont™. The Nafion® allows the transport of hydrogen ions but is an insulator relative to electrons. The anode and cathode layers may be cast directly onto the electrolyte, or extruded and transferred before bonding to the electrolyte.

The humidity level of PEMFCs must be carefully maintained, as too little humidity reduces the transport of ions and can lead to cracking of the membrane, and too much humidity results in condensation which blocks gas flow to reacting sites. Most PEMFCs require humidification of the incoming hydrogen fuel. This water vapor is either supplied through an external source or by recycling the separated vapor from the reacted products and excess air. Some PEMFCs utilize external steam sources and often the inefficiencies associated with this external supply are ignored.

Carbon monoxide acts as a poison to all PEMFCs, and high purity hydrogen is required as the fuel source. As is discussed in section A.2.2, hydrogen is not found naturally and is only available from either the inefficient electrolysis of water or from the reformation of fuels. After reforming fuels, careful separation and/or complete conversion of CO into CO_2 must be made to avoid poisoning of the PEMFCs.

A.3.1.1. PEMFC Applicability

PEMFCs were originally used for early NASA space missions, before being replaced by alkaline systems. In recent years PEMFC reliability, cost, and lifetime have been improved and are now in use with the NASA Space Shuttle Orbiter [54]. PEMFCs have been, and are currently being (2009), demonstrated within experimental cars, city busses (in California), and combined heat and power (CHP) applications [54]. PEMFCs are mostly commonly used for small generation applications of less than 50W and have even been proposed for very small applications such as personal electronics (cell phones). Large PEMFC could be constructed to meet large loads; however, other fuel

cell technologies offer less complex control strategies (maintaining humidity through large stacks becomes more problematic than is desirable).

Many different manufacturers have built PEMFCs, but it is of interest that recently (2009), Ballard and Plug Power have partnered to supply PEMFC powered forklifts to shipping distribution centers. PEMFCs seem ideal for use within interior warehouses due to the benefits of zero pollution and that the forklifts never travel far from an on-site refilling station.

A.3.2. Direct Methanol Fuel Cell Technology

Direct methanol fuel cells (DMFC) are fueled by liquid methanol and operate between 90° and 120°C. The anode reacts with methanol (CH_3OH) and water to form CO_2 and $6H^+$ releasing 6 electrons to the circuit. The hydrogen ions are transported through an electrolyte similar to that used in PEMFCs. Methanol is preferred over hydrogen due to the inherent ease of storing a liquid fuel. While methanol provides substantially smaller energy density (kW/kg) compared to hydrogen, the volumetric energy storage is much higher (kW/m^3)[54]. In space missions weight is the most important factor. However, for use within buildings, volume is the greater consideration.

A.3.2.1. Direct Methanol Fuel Cell Applicability

Unfortunately, DMFC suffer from several inherent internal losses, causing poor voltage current performance as compared to PEMFCs. However, DMFCs have been proposed for use when small amounts of electricity are needed for long periods of time, such as in personal electronics (including cell phones and laptop computers) [54]. Interest in DMFC has decreased as lithium battery technology continues to improve [54].

A.4. Medium Temperature Fuel Cell Technology

Medium temperature fuel cells have startup times and load response between those of low- and high-temperature fuel cells. The moderate temperatures of these fuel cells, between 90° and 260°C, lower both material costs and thermal expansion differences between internal components. However, medium temperature fuel cell exhaust is not

sufficiently hot enough to power turbomachinery, neither as a bottoming cycle nor as a hybrid configuration. Thus, medium temperature fuels cells have not been heavily adopted.

A.4.1. Alkaline Fuel Cell Technology

Alkaline fuel cells (AFCs) were used during the NASA Apollo missions [52]. Due to the AFC's moderate temperatures and electrolyte material, chemical reactions occur more quickly than in PEMFCs and do not require expensive catalytic surfaces [52]. AFCs transport oxygen ions across the alkaline electrolyte. Potassium hydroxide is preferred as an electrolyte due to its low cost, high solubility, and limited corrosivity [54].

Material costs of AFCs are exceptionally low compared to all other fuel cell options and will likely always remain so, due to the materials cost of the electrolyte, anode, and cathode. The system complexity is simpler than that of PEMFCs as humidity controls are not required. However, power density is substantially less than in other options, due to difficulties in manufacturing bipolar plates [54].

AFCs require pure hydrogen fuel and pure oxygen as they are susceptible to CO_2 poisoning at atmospheric concentrations. AFCs operate at temperatures between 90°C and 260°C [54].

A.4.1.1. Alkaline Fuel Cell Applicability

AFCs were used by NASA due to their high reliability and low weight; however, these units were fueled by pure oxygen and hydrogen, thus poisoning was not an issue. AFCs have been proposed for use as regenerative batteries, where excess electricity is used to electrolyze water, producing and separating pure oxygen and pure hydrogen. When electricity is needed, these gases are used to fuel an AFC, recombining to form water [54].

A.4.2. Phosphoric Acid Fuel Cell Technology

Phosphoric acid fuel cells (PAFCs) utilize a phosphoric inorganic acid (H_2PO_4) to conduct protons (hydrogen ions) from anode to cathode. Operating temperatures are limited to 200°C by the chemical stability of the acid. A platinum catalyst is required on both the anode and the cathode. The fuel supply must contain less than 1% CO and less than 50 ppm sulfur, as these contaminants bond to the platinum catalyst, blocking reaction sites [54]. However, the electrolyte may be reactivated by increasing the temperature and/or polarizing the cell at high potentials [54]. PAFCs are resistant to CO_2.

A.4.2.1. Phosphoric Acid Fuel Cell Applicability

PAFCs are the first fuel cells to have been commercialized, with over 65MW having been demonstrated throughout the world [52]. However, the high cost of platinum limits the market to high power quality and reliability applications, such as banks, credit card companies, hospitals and computing facilities [54]. Most units are in the range of 50 to 200kW, however 1MW, 5MW, and 11MW power plants have been built. The largest power plant was built by International Fuel Cells and Toshiba for Tokyo Power [54]. Other companies pursuing PAFC technology include: UTC Fuel Cells (in the US), Mitsubishi Electric (Japan), and Toshiba Corporation (Japan)[52].

A.5. High Temperature Fuel Cell Technology

High temperature fuel cells operate at temperatures between 600° and 1000°C. Often these fuel cells require hours of preheating before use, are slow to respond to load changes, and are very susceptible to thermal stresses within the fuel cell [52, 54]. The high temperatures of these fuel cells do not require expensive catalytic metals to speed the electrochemical reactions. However, the higher temperatures introduce thermal expansion discrepancies between the different materials within a fuel cell (interconnects).

Demonstrated efficiencies of high temperature fuel cells have shown only slight improvement over internal combustion engines. However, the economics improve when

the exhaust from the high temperature fuel cell is used for heating domestic hot water or used toward powering a bottoming cycle, thus producing additional electricity as discussed in section A.6. High temperature fuel cells are best suited for stationary medium or large continuous power applications, as heat retention benefits from a higher volume to surface area ratio.

A.5.1. Solid Oxide Fuel Cell Technology

Solid oxide fuel cells (SOFCs) typically use a yttria-stabilized zirconia (YSZ) ceramic as the electrolyte. The YSZ ceramic conducts oxygen ions when temperatures are above 800°C. Research into new doping agents which are mixed into the YSZ aims for lower operating temperatures. Maximum temperatures are limited to 1000°C by the interconnect materials and the thermal stresses incurred.

Due to the high operating temperatures combined with the formation of H_2O on the fuel side (anode), reformation of simple hydrocarbon fuels occurs internally. Methane (CH_4, the largest component of natural gas) is reformed with H_2O to form CO and H_2. Although CO is able to react with the electrolyte, studies show only the H_2 reacts on the electrolyte, forming H_2O. This H_2O leaves the electrolyte, then reacts with the CO, forming CO_2 and additional H_2. This is known as the water/gas shift reaction. Thus, while CO is a fuel for SOFCs, it does not affect the electrochemical performance, as H_2 is always the reacting species.

The solid ceramic electrolyte does not require humidity controls. It can be used in any orientation, and manufactured in a variety of configurations. Both Siemens™ and Acumentrics™ manufacture SOFCs using tubular stacks versus flat bipolar plates (made by GE™), both of which may be cheaply extruded before baking/sintering the ceramic. Sealing each ceramic tube with an end cap remains a challenge. Either a metal cap is melted to seal an end, thus introducing different thermal expansions, or the ceramic itself is formed into an end cap where manufacturing defects and stress concentrations must be carefully monitored. Recently both Siemens™ and Rolls Royce™ have experimented with flat tube geometries, representing 3-5 small tubes joined and squashed. The flat tube decreases the number of interconnects components, and increases the reacting surface

area. Flat plates may also be cheaply extruded, but suffer from stress concentrations. Significant research has been and is currently being done to increase reliability and decrease costs of manufacturing SOFC stacks. However, the high purity raw materials remain a significant cost unlikely to decrease in the future [76, 81-83]. The SOFC stack represents roughly half the cost of a finished SOFC system. The remainder of the cost consists of the *balance of plant*, the components needed to support the operation of the fuel cell, including blowers, valves, fuel compressors, and high temperature plumbing.

The primary disadvantage of SOFCs is that ceramic is inherently brittle and susceptible to fracture through physical or thermal stress. Care must be taken to avoid large thermal gradients within the stack, which lead to thermal stresses. Thermal gradients occur during operational load fluctuations and when starting and shutting down an SOFC system.

A.5.1.1. SOFC Applicability

Demonstrations of SOFCs have thus far been stationary units, smaller than 300kW. Proposals have been made to power ocean vessels, as power demands are steady and the ship provides minimal physical stress to the unit. Also, a transportable power plant designed for military deployments has been proposed. Benefits of using SOFC technology include the ability to use a range of fuels, whatever the local area provides, and quieter operation. However, it is unlikely to be adopted by the military due to the physical stresses incurred to the fragile SOFC during delivery/transport. Both NASA and Boeing have considered a small SOFC for use within airplanes; however, the physical stress during landings would likely prove too dangerous for flight safety standards [15, 16].

A.5.1.2. SOFC Manufactures

Since 2004, the author has witnessed the SOFC industry deteriorate and begin to crumble. In 2008 GE™ closed their SOFC plant due to the unprofitability of the division. Also Westinghouse™ cut ties with Siemens™, and Siemens™ has scaled back its SOFC manufacturing. Acumentrics™ has made some progress but has yet to cut costs and

increase reliability to commercial standards. The same can be said for Delphi™. Plug Power™ has shown steady negative cash streams on past financial reports, and virtually nothing is known of Bloom Energy™.

A.5.2. Molten Carbonate Fuel Cell Technology

"The electrolyte of a molten carbonate fuel cell (MCFC) is a molten mixture of alkali metal carbonates – usually a binary mixture of lithium and potassium, or lithium and sodium carbonates, which is retained in a ceramic matrix of $LiAlO_2$" [54]. A MCFC must be heated to 650°C for the electrolyte to transfer carbonates (CO_3) from the cathode to the anode. H_2 and/or CO may fuel the anode; however, the cathode must be supplied by both O_2 and CO_2. The necessary CO_2 is often supplied by combusting the exhausted anode gases with the inlet air prior to entering the cathode. This forms the necessary CO_2 along with H_2O and excess air and also serves to preheat the cathode inlet gas mixture.

An operating temperature of 650°C allows internal reformation without an expensive catalysis, but the temperature is low enough to utilize less expensive interconnect components. However, the molten/liquid state of the electrolyte introduces limitations of the system, mainly in the operating orientation and physical stresses associated with the electrolyte-retaining matrix.

A.5.2.1. MCFC Applicability

MCFCs are best suited for stationary combined heat and power (CHP) applications. MCFCs have been considered for use with coal and natural gas resources due to their tolerance of CO [52]. FuelCell Energy (a US company) has pursued MCFC technology and has several partnerships with companies to supply fuel cells, such as Caterpillar Inc. and MTU Friedrichshafen (in Germany).

A.6. Fuel Cell/Gas Turbine Hybrid Technology

Exhaust from both SOFCs and MCFCs (high-temperature fuel cells) are suitable to power turbomachinery, often assisted by an auxiliary burner to increase the turbine inlet

temperature (TIT). The combination of fuel cells and turbomachinery provides a unique synergy. From one point of view, the turbomachinery acts as a bottoming cycle producing additional energy from the fuel cells' exhaust heat. And on the other hand, the fuel cell benefits from the 'free' airflow, and increased pressure provided by the turbomachinery ('free' airflow because electricity is not required to power a fan or blower). However, finding a perfect match between these two technologies has proven difficult. It is relatively straightforward to design a fuel cell hybrid to operating at a single condition (called the design point). However, asking a hybrid to operate over a range of partial loads often upsets the balance between the airflow supplied by the turbomachinery and the cooling needs of the fuel cell. As a result more complex control strategies have incorporated bleed and by-pass values. The effort to correct this discrepancy has had the unfortunate consequence of decreased efficiencies, and increased system complexity and cost.

A.6.1. Fuel Cell/Gas Turbine Hybrid Theoretical Maximum Efficiency

A calculation of the theoretical efficiency limit of the synergy between high temperature fuel cells and turbomachinery further strengthens the appeal of fuel cell/gas turbine hybrids. The first law of thermodynamics represents the conservation of energy (given the symbol ΔH). Simply stated, all energy entering the system must leave the system in one form or another. This means the chemical energy within the fuel is conserved, and will be converted into electrical and heat energy (as well as the chemical energy of new molecules).

The second law describes the conservation/growth of entropy, given by the symbol ΔS (otherwise known as disorder). This law states that the universal entropy may never decrease, and only under ideal conditions remains equal. It is important to remember that entropy may be transferred out of a system, such that the entropy within the system decreases, but the entropy of the universe remains equal to or larger than when the process began. From the second law, Carnot's limit describes the best-case efficiency for a heat engine (such as an internal combustion engine, or steam cycle). However, fuel

cells directly convert chemical energy to electrical energy, and are therefore not heat engines; Carnot's limit, then, does not apply to fuel cells.

However, the third law of thermodynamics does apply to fuel cells and clearly predicts a maximum efficiency. The third law expresses the Gibbs Free Energy, given by the symbol ΔG as shown in Equation A.1. The Gibbs equation predicts how much energy is available to a process, accounting for both the total change in energy (ΔH) and subtracting the energy required for entropy as a function of temperature ($-T\Delta S$). This '$-T\Delta S$' term is the heat energy exhausted by the fuel cell.

Using the third law of thermodynamics the theoretical efficiency of a hydrogen fuel cells is calculated to be 79% as shown by Equation A.1, assuming a water vapor product at 20°C [54]. However, operating inefficiencies, particularly the processing of fuel into hydrogen, causes demonstrated efficiencies to be much lower (often below 30%) [54, 84]. Raising the operating temperature of fuel cells lessens the theoretical efficiency and increases the exhausted heat losses, as expressed by the $-T\Delta S$ term. However, the operating inefficiencies within the fuel cell are greatly lessened at higher temperatures, especially those associated with the reforming of the fuels into hydrogen and as we will see the exhaust heat energy ($-T\Delta S$) can be recovered.

$$\Delta G_{FC} = \Delta H - T_{FC}\Delta S$$

Equation A.1: Gibbs Free Energy: Maximum Energy Available from a Fuel Cell

The high operating temperatures of SOFCs and MCFCs, combined with the exhausted heat energy ($-T\Delta S$, the inefficiencies), supply adequate thermal energy to power a bottoming cycle or turbine. A bottoming cycle is a heat engine, with a maximum efficiency governed by Carnot's limit, as shown in Equation A.2. The bottoming cycle's higher temperature is supplied by the fuel cell's exhaust, while the cold temperature is set by the ambient air conditions. These temperatures are represented by T_{FC} and T_O respectively as shown in Equation A.2.

$$n = (T_{FC} - T_O) / T_{FC}$$

Equation A.2: Carnot's Efficiency Limit of a Heat Engine

The maximum energy recovered by a bottoming cycle, Carnot's limit, when heated by the available exhausted thermal energy from the fuel cell, $-T\Delta S$, results in Equation A.3.

$$(T_{FC} - T_O) / T_{FC} \times T_{FC}\Delta S = (T_{FC} - T_O)\Delta S = T_{FC}\Delta S - T_O\Delta S$$

Equation A.3: Bottoming Cycle Recovered Energy from Fuel Cell Exhaust

Rearranging the Gibbs free energy, from Equation A.1, to represent the Gibbs energies at both the fuel cell and ambient temperatures, results in Equation A.4.

$$T_{FC}\Delta S = \Delta H - \Delta G_{FC}$$
$$T_O\Delta S = \Delta H - \Delta G_O$$

Equation A.4: Gibbs Free Energy at Fuel Cell and Ambient Temperatures

Summing the total energy produced from both the fuel cell, as given in Equation A.1, with the energy recovered by a bottoming cycle, as shown in Equation A.3, results in Equation A.5. Here, upon substitution of Equation A.4 and subsequent cancellations, we find the theoretical limit of a fuel cell with a bottoming cycle to be ΔG_O, which is 79%.

$$= \Delta G_{FC} + T_{FC}\Delta S - T_O\Delta S$$
$$= \Delta G_{FC} + (\Delta H - \Delta G_{FC}) - (\Delta H - \Delta G_O)$$
$$= \Delta G_O$$

Equation A.5: Total Energy Available from a Fuel Cell Hybrid

Thus, it is feasible and desirable to operate high temperature fuel cells, even though the theoretical efficiency is lower at higher temperatures. This is because operating

inefficiencies are lower and a bottoming cycle is capable of recovering the exhausted heat energy. Additionally, all fuel cells require an oxygen source (usually air) blown through the system both for chemical reactions and thermal management. Powering electrical blowers and fans creates system parasitic losses that lower the system's net electrical efficiency. However, a compressor/turbine system can serve as both the supplier of air to the fuel cell and as the bottoming cycle of the hybrid. A quotation from Fuel Cell Systems Explained by Larminie and Dicks summarizes this synergy [54];

> "A fuel cell operating at around 800/900/1000°C can approach the theoretical maximum efficiency. At these temperatures heat engines are also at their best – they do not require exotic materials and are not too expensive to produce. As A.J Appleby has put it:
>
> 'Thus, a high-temperature fuel cell combined with, for example, a steam cycle condensing close to room temperature is a 'perfect' thermodynamic engine. The two components of the perfect engine also have the advantage of practically attainable technologies. The thermodynamic losses (i.e. irreversibilities) in a high-temperature fuel cell are low, and a thermal engine can easily be designed to operate at typical heat source temperatures equal to the operating temperature of a high-temperature fuel cell. Thus, the fuel cell and the thermal engine are complementary devices, and such a combination would be a practical 'ideal black box' (or because of its low environmental impact, a 'green box') energy system.' (Blomen et al., 1993, p. 168). [54, 85]."

A.6.2. Fuel Cell/Gas Turbine Hybrid Classification

Fuel cell/gas turbine hybrids fall into two categories, classified by the placement of the fuel cell, (either between the compressor and turbine, or after the final turbine stage). Such hybrids are classified as either a pressurized/direct hybrid or as an atmospheric/indirect hybrid, respectively. A pressurized hybrid is also referred to as a direct system due to the fuel cell's exhausted heat flowing directly into the turbine, whereas an atmospheric hybrid uses a heat exchanger to indirectly transfer the exhausted heat into the pressurized air before powering the turbines.

Pressurizing a fuel cell adds control complexity, in that the fuel supply must also be able to meet the same pressures supplied by the turbomachinery. However, the fuel cell benefits from increased pressure, due to the perceived increase in fuel cell electrolyte area. Basically, higher pressures cause more molecular collisions with the electrolyte surface, thus increasing reactions and producing more electricity, as predicted by the Nernst equations.

Both direct and indirect hybrids utilize a heat exchanger (which recovers heat energy otherwise exhausted) to preheat the compressed air. However, heat exchangers can only transfer a fraction of the thermal energy from the exhaust to pressurized air, and also represent a significant expense of any hybrid system. As such, a pressurized/direct hybrid shows performance and cost benefits over that of an indirect hybrid.

A.7. Fuel Cells Meeting Electrical Demands

Most electrical loads fluctuate on a daily, weekly, or yearly cycle driven by many factors, such as business hours, heating and/or cooling needs, and consumer demand. Thus, it is important that a power plant be able to supply these changing loads without sacrificing efficiency. As new sources of power are introduced to the electrical grid, such as wind and solar, new transients are introduced and must be met and stabilized, even down to the millisecond time scale.

Fuel cells are of great interest due to their high electrical efficiencies and low pollutants; however, their transient response is largely unproven. Many research groups have modeled transients, as discussed in section 2.3. While modeled results are promising, fuel cell manufacturers remain hesitant to push demonstrated fuel cells to such limits. The predominate groups attempting this include the National Fuel Cell Research Center (NFCRC) at the University of California, Irvine (UCI), the National Energy Technology Laboratory (NETL), and the Thermochemical Power Group (TPG) at the University of Genoa, Italy.

Fuel cells that have been demonstrated minimize the number of system shutdowns, due to the thermal and chemical shock experienced by the system. Even demonstrated PEMFCs powering city buses are started only once a day, and a large and expensive

battery pack is used to assist the transient power demands of the bus. Systems demonstrated to date have operated over a narrow range centered about the design point. The proposed system shows operation of a very wide range, and could make startup and shutdown less stressing to the system.

Areas targeted for improving fuel cell transient response include tightly regulating the fuel cell thermal requirements, minimizing gas plenum volumes and plumbing, and using simple turbomachinery with short shafts [2]. The system presented within this thesis improves upon these areas as compared to previously demonstrated and modeled systems. However, transient analysis has yet to be performed on the proposed system.

A.8. SECA (Solid State Energy Conversion Alliance)

SECA is a department of energy (DOE) project headed by the National Energy Technology Laboratory (NETL) with the goal of furthering the commercialization of fuel cell technology, primarily SOFCs [86-91]. The SECA program has set several goals for participants to achieve, in order to continue to the next round of funding.

APPENDIX B: Modeling Platform Overview

B.1. Modeling Platforms

Models are developed using specialized computer codes and run from a modeling platform. The modeling platform should be chosen carefully, as all future work and results will come from this fundamental decision. An appropriate modeling platform must include built-in functions and solvers, the ability to write custom code, the ability to save and output data for later use, and the ability to share the program code with others. Due to its popularity and modeling power, Matlab®/Simulink® is the modeling platform used by the UAF model as presented in this thesis.

The UAF model is built to study the steady-state performance of the proposed system (and as a toolkit for other configurations). While not yet incorporated, consideration has been made for inclusion of micro-level dynamics, computational fluid dynamics (CFD) and/or finite element method (FEM) modeling to be later included if so desired. Many research groups world wide have chosen to use the Matlab®/Simulink® modeling environment including the University of California, Irvine (UCI), the National Energy Technology Laboratory (NETL), and the University of Genoa, Italy's Thermochemical Power Group (TPG) [92]. NETL has incorporated both CFD and FEM analysis into their Matlab® simulations. Below is a summary of other modeling platforms.

B.1.1. Microsoft Excel®

Microsoft Excel® is widely known as a spreadsheet application. However, its power greatly increases when using both the built-in solver and custom written macros. Excel® was initially chosen for the UAF model based on the prior and current work of Dr. Thomas Wolf of Brayton Energy [51]. Dr. Wolf has spent many years of his professional career fine-tuning his Excel® modeling skills, and uses his expert intuition on turbine behavior to justify assumptions without over-simplifying the problem. The author is continually amazed at the power of Excel® when solving difficult problems. However, the author believes that as the system becomes increasingly complex, and possibly

transient (time dependent), it will become much more difficult to work with Excel® as a modeling platform.

B.1.2. Matlab®/Simulink®

Matlab® is a powerful and popular computation package that specializes on matrix and vector mathematics while offering a very intuitive code writing environment. Simulink® is built on top of Matlab®, and specializes in solving transient systems. Simulink® uses a graphical programming interface of connecting-the-blocks to direct outputs of one function to become inputs of another. This offers a visually intuitive interface, similar to that of Labview®. (Examples of the Simulink® interface are shown in Figure 3.1.) Simulink® has access to Matlab's® vast collection of built-in functions, add-on toolboxes, and custom code. Additionally, Matlab®, and therefore also Simulink®, can communicate with *C*, C++, Python, and many other programming languages.

Critics of Matlab® know that the user-written custom code (called M-files) when run are interpreted, as opposed to (faster) compiled languages such as C and C++. Matlab®/Simulink® is the modeling platform used by the National Fuel Cell Research Center (NFCRC) at the University of California at Irvine (UCI) [92], the Thermochemical Power Group (TPG) at the University of Genoa, Italy [45], National Energy Technology Laboratory (NETL) in Morgantown, West Virginia[32, 33], and many other researchers around the world.

B.1.3. Homer®

Homer® was developed by the National Renewable Energy Laboratory (NREL) [93], and is free to download and use. Homer® is intended to evaluate the economic results of meeting thermal and electrical loads by considering different combinations of storage and generation options. Homer® is most useful when considering the addition of new equipment to existing systems. However, it is also useful in evaluating the economic feasibility of installing renewable options at remote/distributed generation sites. The Homer® model requires inputs for the considered energy resources (such as

diesel engines, wind power, hydro, solar, batteries), the costs of fuel, value of electricity and lastly, the electrical and thermal loads for every hour throughout a yearly cycle. Results depend heavily on the accuracy of user data for both solar and wind resources, but estimates may be made based on monthly averages of wind speeds, and solar resources based on longitudinal location and average cloud cover. Off-design efficiency of diesel generators and wind turbines are predicted based upon performance maps of each technology.

Homer® calculates how the electric and thermal loads will be met for every hour of a yearly cycle for each of the proposed cases, consisting of different combinations of system components and economic sensitivities. Homer® considers capital costs, operation and maintenance (O&M) costs, and replacement costs for each component of the system. Also, a given annual interest rate and the lifetime of the project allow determination of when particular options will become economically viable (which requires sensitivity analysis into fuel costs, interest rates, and even scaling of wind speeds [93]).

Homer® is designed to answer an economics question, and is not able to consider the thermodynamics of a hybrid engine, where components such as a fuel cell and gas turbine share heat and gas flows. Unfortunately, Homer® was unable to assist in the economical study of this thesis work due to the limitation that when computing the off-design efficiency of engines, Homer® assumes a linear fuel curve that must pass through zero. However, Homer's® graphical presentation of its results was replicated in the economic study, as presented in chapter 5.

B.1.4. RETScreen®

RETScreen® is another economical energy modeling tool, developed in Canada. However, RETScreen® does not study how the system performs at an hourly level; rather it calculates the economics assuming the configured system is able to meet all loads. RETScreen® is run within Excel®, making the program easily available. The author has little experience using RETScreen®, as Homer® was preferred. However, a

consensus was reached that results found using Homer® should be used as the inputs into RETScreen®, thus gaining more advanced economical analysis over Homer®.

B.1.5. Others

This thesis focuses on modeling the broad performance of system components. However, consideration was made for future inclusion of microstructure and/or fluid dynamics analysis within each component (e.g. the fuel cell stack or compressor /turbine vane geometry). Finite element method (FEM) and computational fluid dynamic (CFD) packages are very useful in studying processes such as fuel mixing, combustion stabilization, temperature gradients, and fuel cell electrolyte gas transport. Modeling packages such as Fluent®, Abaqus®, Comsol® and others could be used for such analysis. NETL demonstrated such coupling within their model when the author toured their facility [94]. Incorporating FEM or CFD analysis will greatly increase the required computation and likely require super-computing or parallelized computation.

APPENDIX C: Code Documentation

C.1. Jargon

An effort is made to use consistent jargon throughout this thesis, as defined below:

M-file: A Matlab® based code, script, or function, only assessable from the Matlab® environment.

S-function: An M-file wrapped in a special Simulink® header, that allows Simulink® access to the Matlab code.

Component: A component refers to a single piece of the system, such as a single compressor, turbine, heat exchanger, or fuel cell. One S-function is written for each component of the system, where inputs and outputs define what gas is entering or leaving the component.

Module: Very similar to a component and typically refers to the fuel cell. The fuel cell module is made of several components packaged together. Thus, the fuel cell module is made up of a heat exchanger component, burner component and fuel cell stack component.

Variable: An input into a component that changes nearly every iteration, such as the temperature, pressure, or mass flow of the gas entering or exiting a component.

Parameter: A value required by a component that does not change in the off-design analysis, such as the compressor and turbine performance map-stretching parameters, the heat exchanger effectiveness and the fuel cell electrolyte area.

Design Point (DP) Parameters: The values assigned to components during the design point simulation, such that the off-design parameters can be found to satisfy the design point's prescribed parameters.

States: States are the final converged variables of each component.

C.2. Simulink® versus Matlab®

The previous Master's work concluded with the coding of system components using S-function in a Simulink® environment. Simulink® was chosen for the intuitively visual

representation of the modeled system. (Figure 3.1 is a cleaned-up version of the UAF model's Simulink® interface.) However, as the system became increasingly complex, Simulink® failed to converge. To gain control over the convergence of the proposed system, the model was moved into the Matlab® environment.

C.3. Simulink® S-Functions

Simulink® gains access to all of Matlab's® functionality through the use of S-functions. An S-function is a special header on an M-file that informs Simulink® how many input and output variables are required, whether there are continuous or discrete states, and allows Simulink® to run the code. The S-function header is directed by "flags" to perform different operations. New flags were defined that allowed the S-funtions to be accessible from either the Simulink® or Matlab® environment. Listed below are the S-function flags used in the UAF model.

Flag 0: The initialization process is where Simulink® requests the number of inputs and outputs of the function. The number of transient states can also be defined with this flag.

Flag 1: The derivatives function is not used within the UAF model.

Flag 2: The transient states are updated with this flag. However, the steady-state UAF model does not use this flag.

Flag 3: The outputs function is where the components' code is run.

Flag 4: Defines when Simulink® must schedule the S-function to run again. The UAF model does not use this flag.

Flag 9: Terminates the simulation, and cleans up system variables.

Custom-created flags include the following:

Flag 10: An old flag being phased out.

Flag 30: Runs the main component code from the Matlab® environment.

Flag 99: Flag 99 terminates the UAF model from the Matlab® environment and saves all component states.

C.4. Transient Modeling

The UAF model has not incorporated any transient code. However, if the model were to be made transient, the S-function flags 1, 2 and 4 (derivatives, updates, and time of next hit) would need to be incorporated by adding transient code to the component S-functions, such as a compressor spool shaft momentum, where the work difference between each turbine and compressor produces a torque and accelerates the entire shaft. The inclusion of transient plenum volumes would be much more involved. However, Cantera has this functionality built-in through its reactor network library. Matlab also has the ability to communicate with a wide variety of FEM and CFD programs if more detailed analysis of components is desired.

C.5. Cantera

Cantera is a toolkit developed by Dr. David Goodwin of Cal Tech for solving problems involving chemical kinetics, thermodynamics, and transport processes [71]. In the previous Master's work, Shomate polynomials were used to compute thermodynamic properties, and solve a custom-coded chemical equilibrium function (Gibbs Energy minimization). However, Cantera achieves chemical equilibrium much more quickly, efficiently, and has the ability to easily consider additional species. Cantera is now incorporated into the UAF model and allows the model to easily consider additional species, and to allow the modeling of time dependent chemical reaction kinetics.

C.6. XTPW – Mass Flow, Temperature, Pressure, and Global Index

The primary inlet and outlet variable to all model components is the XTPW vector, where X represents the total mass flow of the gas mixture (kg/s). T is the gas temperature (K), P is the total gas pressure (atmospheres), and W represents a global index where the Cantera gas species mixture is stored. (Each component/state must be assigned a unique W (WISE) index.) When an XTPW vector is received as an input, the receiving components can look up the particular gas object and species concentration, as referenced by W. Using a global index of Cantera gas objects allows faster computation than the methods used during the Master's work. A gas object for each state within the

system is created only once during the simulation by the initialization flag. The model then modifies each gas object as a solution converges.

C.7. Saving State Information

The Basic Ten:

Detailed values are reported when a solution is reached and flag 99 is run. Every component reports the following basic ten values, and often several other values particular to the component, as is discussed in section C.9. Saving state values allows verification of conservation of mass, enthalpy and entropy creation throughout the system after modeling is complete.

1-4: The inlet XTPW vector to the component

5-8: The outlet XTPW vector of the component

9: The exiting gas enthalpy (kJ/kg) of the component

10: The exiting gas entropy kJ/(kg K) of the component

Species States:

Certain components also report the species concentrations within a state. This vector includes the XTPW of the state outlet plus the mass flow (kg/s) of every species being considered.

C.8. Design Point versus Off-Design Modeling

At the beginning of each simulation the model must be run with the design point (DP) signal on (flag 30 with CANTERA.DP equal to 1). When the DP signal is on, the researcher prescribes conditions such as the inlet mass flow, pressure ratios of each compressor, fuel cell temperatures and voltage, the turbine inlet temperature (TIT), and nozzle settings. This signals the model to solve the appropriate component performance parameters; such that the design point matches prescribed values. These parameters are then used for all off-design modeling.

Off-design modeling is run with flag 30 and CANTERA.DP=0. Even the design point condition is run with the off-design flag, where converged results differ less than 0.001% from design point.

C.9. The Code

The following subsections explain the UAF model's code. A copy of the code can be found in the appendix.

Default inputs used by Simulink® with S-functions include *t*, *x*, *u*, and *flag*. Simulink's® time is reported through *t*, Simulink's® states are saved and reported through *x*, input variables are reported through *u*, and the different flag operations are reported through *flag*. Both *t* and *x* are unused in the final UAF model, however *u* and *flag* are used to input variables and save component states respectively.

C.9.1. Assumptions

The assumptions function (assump.m) is a precursor script used to load data, (including the compressor/turbine performance data), run the global values function and set certain flags to their default status. The CANTERA structure is declared global, and CANTERA.GAS is assigned as *basic13*. *Basic13* is a list of 13 gas species for the model to consider. The 13 gas species are: CO, CO_2, H_2, H_2O, N_2, O_2, Ar, CH_4, C_2H_6, C_3H_8, $C_{(g)}$, NO, and NO_2.

C.9.2. Global Values

Global Values (Global_Values.m) fills in the Cantera global structure with data that will be used throughout the model.

Global Values creates two gas objects named AIR and FUEL, where these specify the gas ambient conditions and composition that are used as the air and fuel inputs. Air is assumed at 15°C, one atmosphere (atm) and 78% N_2, 21% O_2, and 1% Ar by mole or volume fractions. Fuel is assumed pure methane (CH_4) at 25°C and one atm. The AIR and FUEL gas object may be modified to model changing ambient conditions, such as the daily temperature and humidity fluctuations.

The lower heating value of each species is also recorded, along with the moles of O_2 desired for complete combustion, and the electrons that may be contributed to the reaction. This allows the fuel and air utilization to be easily calculated in the fuel cell component. The number and type of atoms making up each species is also recorded, thus allowing a conservation of elements calculation to be made for the chemically reacting components. Many other reference values are saved, such as the number of species, kg/mol and mols/kg of each species, and the vector index of each species.

C.9.3. Gas Inlet

sfun_Gas_Input(t,x,u,flag,Gas_Type,Gas_Flow,WISE, Mf_T_P)

Code Notes:

The gas input component is used to introduce gas (either air, fuel or a custom composition) into the system. The gas source may be specified in one of three different methods:

1st: Mass flow of individual species is specified along with the gas mixtures temperature and total pressure.

2nd: The default air or fuel may be specified with a specified mass flow, where temperature, pressure and composition are determined from the global AIR and FUEL gas object. When the global object is updated, this input will change accordingly; thus changes to the ambient air or natural gas supply can quickly and uniformly be modified throughout the system.

3rd: The composition, temperature, and pressure may be determined from any existing gas object with a preexisting W index; only the mass flow needs to be specified.

Design Point Parameters:

Gas_Type: The method used to define the gas. 0: assigning individual species, 1: global ambient air, 2: global fuel, 3: defined from preexisting gas object.

Gas_Flow: Number specifying the total mass flow of the mixture in kg/s.

WISE: The W global gas index where the gas object will be stored.

Mf_T_P: A vector specifying the mass flows of the individual species in kg/s, the mixture temperature in K, and total pressure in atm. (This parameter is needed only if Gas_Type equals 0 where the individual species are defined).

Off-Design Parameters:

Same as design point parameters.

Inlet Variables:

None.

Outlet Variables:

XTPW

Saved States:

[0,0,0,0, X,T,P,W,H,S] The first four saved states are zeros because the gas input function has no input.

C.9.4. Filter

sfun_Losses(t,x,u,flag,Ploss,Tloss,WISE)

Code Notes:

The inlet filter simply causes a pressure drop and/or temperature losses. Temperature losses are never used within the UAF model.

Design Point Parameters:

Ploss is a decimal representing the fractional pressure loss experienced across the component.

Tloss is the fractional temperature loss across the component.

WISE is the global gas object index for the component exit.

Off-Design Parameters:

Same as the design point parameters.

Inlet Variables:

XTPW

Outlet Variables:

XTPW

Saved States:

None.

C.9.5. Compressor

sfun_Compressor_Lookup(t,x,u,flag, WISE, map, PR_DP, MachSpeed_DP, SurgeMargin_DP, Eff_DP, ShaftSpeed_X, Phi_X, PR_X, Eff_X)

Code Notes:

Based upon the inlet corrected mass flow (*phi*) and the shaft speed, the compressor mach speed, pressure ratio and efficiency are found from the interpolated compressor performance data. The outlet pressure is calculated from the inlet pressure multiplied by the pressure ratio. The outlet temperature is found from first assuming isentropic compression, then adding the inefficiency penalty. The gas composition and mass flow remain unchanged across the compressor. The required compressor work will become an input variable to the connected turbine.

The compressor performance map is a data structure containing both the raw data from the original imported compressor map, and the interpolated parameterized *Beta* values. The *Beta* values contain the following arrays: *speed values* contains the different mach speeds for which data has been parameterized, *Phi* and *PR* are separate arrays for which a fifth degree polynomial is fit to describe the corrected mass flow and pressure ratio respectively at each mach speed value, *Eff* is an array of second degree polynomials describing the efficiency at each mach speed. A curve fitting each parameterized value with polynomials was chosen primarily due to the efficient ability to algebraically solve for the roots (using the Matlab's® roots function). The parameterized polynomials offer a very smooth interpolation of the compressor performance data.

Design Point Parameters:

WISE: The unique global gas object index storing the outlet components.

map: A data structure containing the performance data of the specific compressor.

PR_DP: The prescribed pressure ratio at design point.

MachSpeed_DP: The prescribed compressor mach speed at design point.

Eff_DP: The prescribed compressor efficiency at design point.

SurgeMargin_DP: The desired distance from the surge line. (Should be changed to Beta_DP, as the distance is calculated from beta and not surge margin.)

Off-Design Parameters:

ShaftSpeed_X: A scaling factor applied to the incoming shaft speed, required to allow the compressor to meet its prescribed design point mach speed.

Phi_X: A scaling factor to the phi axis of the compressor map, based upon the SurgeMargin_DP, and the prescribed system airflow.

PR_X: A scaling factor to the pressure ratio axis of the compressor map, based upon the requested PR_DP, MachSpeed_DP and SurgeMargin_DP.

Eff_X: A scaling factor to the efficiency of the compressor map. (Note that in this research Eff_X equals one. No efficiency scaling was required.)

Inlet Variables:

XTPW

The compressor shaft speed as a decimal percentage.

Outlet Variables:

XTPW

Compressor Work required

Isentropic Efficiency

Saved States:

The basic ten and;

11: the compressor efficiency as a decimal;

12: the work (kW) required to perform the compression;

13: the compressor shaft speed as a percentage;

14: the beta value;

15: the true surge margin.

C.9.6. Intercooler

sfun_Intercooler(t,x,u,flag,dT,ploss,WISE)

Code Notes:

The intercooler is a very simplistic heat exchanger. The exit temperature is prescribed as a value above the ambient air temperature. In the case of this study the value was 3.9K above the ambient air temperature of 20°C; thus the intercooler cools the compressed air to 23.9°C. A fixed percentile pressure loss is also considered.

Note that a prescribed blower parasitic loss is assumed throughout the study, such that the dT can always be met.

Design Point Parameters:

dT: Represents the temperature above ambient conditions of the exit gas.

Ploss: Represents the percentile pressure loss.

WISE: The global Cantera gas object index for the working gas exit.

Off-Design Parameters:

Same as the design point parameters.

Inlet Variables:

XTPW of the working fluid (the compressed air).

XTPW of the ambient cooling air.

Outlet Variables:

XTPW of the working fluid.

Saved States:

The basic ten and

11: dT (K)

12: The removed enthalpy, dH (kW/s), from the working gas.

13-15: Are zeros.

C.9.7. Heat Exchanger

sfun_HeatExchanger(t,x,u,flag, n_HX, ploss_hot, ploss_cold, WISE_hot, WISE_cold, TCo)

Code Notes:

The heat exchanger code assumes a prescribed effectiveness and uses the number of transfer units (NTU) method, to calculate the enthalpy transferred from one gas stream to another. It is assumed that the heat exchanger effectiveness remains constant throughout the study.

In future studies it will be possible to make the effectiveness a function of the heat exchanger geometry and the gas flows.

Design Point Parameters:

n_HX: The heat exchanger effectiveness as a decimal (between 0 and 1).

Ploss_hot: The percentile pressure loss experienced by the hot (exhaust) gas stream.

Ploss_cold: The percentile pressure loss for the cold (compressed) gas stream.

WISE_hot: The global Cantera gas object index for the hot (exhaust side) exit stream.

WISE_cold: The global Cantera index for the cold (compressed side) exit stream.

(Note; Tco is currently unused, but once represented an initial guess for the cold side exit temperature.)

Off-Design Parameters:

Same as design point parameters.

Inlet Variables:

XTPW on the hot side (the exhaust products side)

XTPW on the cold side (the compressed air side)

Outlet Variables:

XTPW of the hot side exit

XTPW of the cold side exit

Saved States:

Two states are saved, one for the hot side and one for the cold side, each with the same pattern.

The basic ten,

11: The heat exchanger effectiveness.

12: The change in enthalpy from inlet to exit.

13-15: Are zeros.

C.9.8. Solid Oxide Fuel Cell

sfun_Fuel_Cell(t,x,u,flag, FC_Type, n_Fuel, rec, Ploss_anode, Ploss_cathode, Ploss_combustor, Ploss_HX, FC_Volt, HX_Eff, CellArea, CellTemp, FCExitTemp, WISEAirHX, WISEAirFC, WISEPComb, WISEPHX, WISEFuelReform, WISEFuelFC, WISErec1, WISErec2)

Code Notes:

The primary goal of the fuel cell code is to retain the cathode exit temperature at a prescribed value, which is accomplished using an internal solver. The code assumes a fixed percentile anode recirculation, and fixed internal heat exchanger effectiveness.

Design Point Parameters:

FC_Type: Defines the type of fuel cell to be studied. 1 represents an SOFC and is the only fuel cell coded. However, other transport mechanisms could be written and assigned new numbers, allowing the system to be easily configured for different fuel cells.

n_Fuel: The prescribed fuel utilization across the fuel cell anode as a decimal.

rec: The prescribed anode recirculation as a decimal.

Ploss_anode: The percentile pressure loss across the anode stack as a decimal.

Ploss_cathode: The percentile pressure loss across the cathode stack as a decimal.

Ploss_combustor: The percentile pressure loss across the combustor as a decimal.

Ploss_HX: The percentile pressure loss across the heat exchanger as a decimal.

FC_Volt: The desired fuel cell operating voltage of each stack.

CellTemp: The desired stack exit temperature.

FCExitTemp: The desired fuel cell module exit temperature (after post-combustion and pre-heating the inlet cathode air).

WISEAirHX: The global Cantera gas object index for the air exiting the internal heat exchanger, before entering the cathode.

WISEAirFC: The index for the air exiting the cathode.

WISEPComb: The index for the products exiting the post combustion.

WISEPHX: The index for the products exiting the internal heat exchanger, after having preheated the inlet cathode air.

WISEFuelReform: The index for the partially reformed fuel before entering the anode.

WISEFuelFC: The index for the products exiting the anode.

WISErec1: The index for the anode exit products to be recycled to the inlet.

WISErec2: Currently unused; however, could be used if a recycle compressor or ejector is modeled, to represent an intermediate point in the recycle trip, between the anode exit and the WISEFuelReform point.

Off-Design Parameters:

The above design point parameters define the following:

HX_Eff: The internal heat exchanger effectiveness, such that both the stack exit temperature and the fuel cell module exit temperature meet the design point requirements.

CellArea: The fuel cell stack area, such that the current density is satisfied to meet the prescribed design point fuel cell voltage.

Values of FC_Volt and FCExitTemp are unused during the off design study.

Values of n_Fuel and rec could be changed at will during the off-design study.

Inlet Variables:

XTPW of the fuel.

XTPW of the air supply.

Outlet Variables:

XTPW vectors are reported for the following states within the fuel cell module.

The preheated inlet air.

The post-combustion products.

The products leaving the internal heat exchanger and exiting the fuel cell module. (The most important XTPW vector, as this must be connected to the next component.)

The pre-reformed fuel before entering the anode.

The anode exit.

At recycle 1 state.

At recycle 2 state.

The fuel flow required is reported.

The work (kW/s) produced from the fuel cell.

The fuel cell voltage.

The fuel cell average current density.

Saved States:

Several states are saved for the fuel cell module as listed below.

Fuel cell parameters: rec; n_Fuel; HX_Eff; Ploss_anode; Ploss_cathode; Ploss_combustor; Ploss_HX; JFC; VFC; Vo; dVJ_J; dVT_J; dVP; dVoxygen; dVfuel. Where VFC is the cell voltage, Vo is the predicted maximum voltage before losses, dVJ_J is the IR, current losses, dVT_J is the loss associated with cell temperature, dVP is the voltage loss associated with the operating pressure of the cell, dVoxygen is the oxygen utilization loss (Nernst oxygen concentration), dVfuel is the fuel utilization loss (the Nernst hydrogen/H_2O concentration).

The air leaving the internal heat exchanger uses the basic ten plus the enthalpy difference received from the heat exchanger.

The air leaving the fuel cell, uses the basic ten plus the cell voltage, cell current density, and fuel cell work.

The post combustion products are just the basic ten.

The products exiting the internal heat exchanger (and about to exit the fuel cell module) include the basic ten and the enthalpy differnce lost to the heat exchanger.

The pre-reformed fuel (before entering the anode) reports just the basic ten.

The anode exit include the basic ten and the cell voltage, cell current density, and fuel cell work.

The recycle 1 and 2 streams are just the basic ten.

The fuel in and air in (before pre-reforming or pre-heating) are the basic ten with the first four values set to zero.

Species states are reported for the pre-heated air, the inlet fuel, the pre-reformed fuel, the anode exit, the cathode exit, the post-combustion, and the products leaving the internal heat exchanger.

C.9.9. Combustor

sfun_Combustor(t,x,u,flag,Tout, hloss, Ploss, WISE)

Code Notes:

The combustor code solves for the fuel required to meet a prescribed temperature, unless the inlet gas temperature is already greater than the prescribed value.

Design Point Parameters:

Tout: The prescribed outlet temperature.

hloss: The energy losses associated with incomplete combustion and heat loss, expressed as a decimal.

Ploss: The percentile pressure loss as a decimal.

Off-Design Parameters:

Same as the design point parameters.

Inlet Variables:

XTPW of the air (or fuel cell exhaust)

XTPW of the fuel supply. (Note that while X is the mass flow of the inlet fuel, the combustor code will solve for what is actually needed.)

Outlet Variables:

XTPW of the exhaust products.

The mass flow of fuel required (kg/s).

Saved States:

The basic ten for the air inlet and products outlet

11-14: An XTPW vector for the fuel inlet.

Species states are saved for the air inlet, the fuel inlet, and the products outlet.

C.9.10. Turbine

sfun_Turbine_Lookup(t,x,u,flag, WISE, map, ShaftSpeed_DP, MachSpeed_DP, Eff_DP, V_X, Phi_X, ShaftSpeed_X)

Code Notes:

Turbine performance maps are fitted using the following equation: Phi=1-exp(-k*(ER-1)), where ER is the expansion ratio and k is the interpolated value for different mach speeds. Efficiency is found from the U/Co approximation, where U is the mean tip speed of the turbine and Co is the square root of 2 times the isentropic enthalpy difference.

Design Point Parameters:

WISE: The unique global gas object index for the outlet stream.

map: A data structure defining the performance of the turbine.

ShaftSpeed_DP: The prescribed shaft speed percentage at design point.

MachSpeed_DP: The prescribed mach speed of the turbine at design point.

Eff_DP: The prescribed efficiency of the turbine at the design point. (Note that it is assumed that peak efficiency also occurs at design point.)

Off-Design Parameters:

V_X: A scaling multiplier applied to the average tip speed, used to adjust the peak efficiency to the design point.

Phi_X: A scaling factor applied to the corrected mass flow axis.

ShaftSpeed_X: A scaling factor applied to relate the mach speed to the shaft speed.

Inlet Variables:

XTPW and the shaft work (kW) that must be produced accounting for the compressor and shaft bearing losses.

Outlet Variables:

XTPW, the turbine efficiency, the U/Co value, and the shaft speed percentage.

Saved States:

The basic 10 and;

11: The turbine efficiency

12: The Work (kW) produced by the turbine

13: The shaft speed percentage

14: U/Co

C.9.11. Variable-Geometry Turbine

sfun_Var_Turbine(t,x,u,flag, WISE, map, ShaftSpeed_DP, MachSpeed_DP, Eff_DP, V_X, Phi_X, ShaftSpeed_X, Constraint)

Code Notes:

The variable geometry turbine code very closely represents the normal turbine code, with the differences that it is constrained to operate at an expansion ratio that exhausts at atmospheric pressure, and at another set condition. In this study the other condition is the turbine optimal efficiency, but could also be a fixed shaft speed for synchronous generators, or constant mach speeds for a supersonic shockwave turbine. The variable nozzle settings scale the corrected mass flow axis of the turbine performance map (as a multiplied factor).

Design Point Parameters:

WISE: The global Cantera gas object index for the outlet products.

map: The data used for interpolating the turbine performance.

ShaftSpeed_DP: The prescribed design point (DP) shaft speed as a decimal.

MachSpeed_DP: The prescribed mach speed line at which to operate at the design point.

Eff_DP: The prescribed efficiency to normalize the U/Co efficiency constraint.

Constraint: The constraint used for the turbine, either at highest efficiency, fixed shaft speed, or fixed mach speed.

Off-Design Parameters:

The above design point parameters solve for the following:

V_X: The U/Co multiplication factor needed to meet the prescribed design point condition.

Phi_X: A scaling factor for the corrected mass flow axis.

ShaftSpeed_X: A scaling factor to meet the design point condition between shaft speed and mach speed.

Inlet Variables:

XTPW of the inlet gas stream.

The outlet pressure (atm), such that the products will exhaust to ambient conditions.

The variable nozzle setting.

Outlet Variables:

XTPW of the outlet stream.

The work (kW/s) produced by the turbine.

The mass flow needed for the turbine to operate at its constraint.

The turbine efficiency.

The turbine U/Co, used for calculating the turbine efficiency.

And the turbine shaft speed.

Saved States:

The Basic ten plus,

11: The turbine efficiency.

12: The produced turbine work (kW/s).

13: The shaft speed.

14: The U/Co value.

15: The nozzle setting.

16: The gas flow (kg/s) needed to meet the constraint.

APPENDIX D: Code Dump

D.1. Assumptions

/Users/winstonburbank/Documents/Matlab Work/PHD Library/assump.m 1 of 1

```
 1 %Assump.m   Defaults For Esystems
 2 global CANTERA
 3 load HP_Compressor.mat, load LP_Compressor.mat, load Generic_Turbine.mat
 4 load RR_LPC.mat,  load RR_HPC.mat,
 5 % load Engine,
 6
 7
 8 CANTERA.GAS = 'basic13';
 9
10 run GLOBAL_VALUES
11
12 CANTERA.DP = 0;
13 CANTERA.DATA = 1;
14 CANTERA.PLOT = 0;
15 CANTERA.ERROR= 0;
16
```

D.2. Global Values

/Users/winstonburbank/Documents/Matlab Work/PHD Lib.../GLOBAL_VALUES.m 1 of 5

```
 1 % GLOBAL_VALUES.m
 2 %  Global Variables Used in Esystems
 3 %  HV Heating values are lowwer heating values in (kJ/kg)
 4 function []=GLOBAL_VALUES()
 5 global CANTERA
 6
 7 gas= importPhase('Basic.cti',CANTERA.GAS); %temp var only in this function
 8
 9 % CANTERA.ROUNDING=6; %number of decimal to keep when rounding off
10 % CANTERA.TMIN=273;
11
12 CANTERA.AIR= importPhase('Basic.cti',CANTERA.GAS);
13 set(CANTERA.AIR,'T',15+273.15,'P',oneatm,'X','O2:0.21, N2:0.78, AR:0.01');
14
15 CANTERA.FUEL= importPhase('Basic.cti',CANTERA.GAS);
16 set(CANTERA.FUEL,'T',25+273.15,'P',oneatm,'X','CH4:1');
17
18 CANTERA.REF.T = 25 + 273.15;  CANTERA.REF.P = 1 * oneatm; CANTERA.REF.R =↙
8.31447; % m3 Pa / (K mol)
19 CANTERA.LIMIT.T=[-10,3000]+273.15; CANTERA.LIMIT.P=[oneatm/10, 50*oneatm];
20 CANTERA.NSP=nSpecies(gas);
21 CANTERA.KG_KMOL = molarMasses(gas);
22 CANTERA.KMOL_KG = 1 ./ CANTERA.KG_KMOL;
23
24 %% species index
25 CANTERA.I=null(1,1);
26 vco2=zeros(CANTERA.NSP,1); vco2(speciesIndex(gas,'CO2'))=1;
27 vh2o=zeros(CANTERA.NSP,1); vh2o(speciesIndex(gas,'H2O'))=1;
28 vo2 =zeros(CANTERA.NSP,1); vo2(speciesIndex(gas,'O2'))=1;
29 vn2 =zeros(CANTERA.NSP,1); vn2(speciesIndex(gas,'N2'))=1;
30
31 CANTERA.Hf=zeros(CANTERA.NSP,1); %Enthalpy of Formation at 298.15 K:  Used to↙
get heating values
32 CANTERA.BC=zeros(CANTERA.NSP,CANTERA.NSP); %Basic/Complete combustion↙
rules:  Used to get heating values
33 CANTERA.O2_NEEDED = zeros(CANTERA.NSP,1); %Oxygen Needed in a stream for↙
complete combustion
34 CANTERA.ELECTRONS = zeros(CANTERA.NSP,1); %Electrons released when gas is↙
transfered through an electrolyte.
35 CANTERA.FUELSTREAMS = zeros(CANTERA.NSP,1);%logical list of what can be used↙
as a fuel
36 CANTERA.ELEM = zeros(nElements(gas),CANTERA.NSP);%Will Count the number of↙
atoms
37 CANTERA.HV=zeros(CANTERA.NSP,1); %Gas Heating Value kJ/kg
38 for i=1:CANTERA.NSP
39     name=char(speciesName(gas,i));
40     z=zeros(1,CANTERA.NSP)';
```

```
41  switch name
42    case 'CO',
43      CANTERA.I.CO=i;
44      CANTERA.Hf(i)=-110.53;   %CO +-0.17 kJ/mol
45      z(i)=1;
46      CANTERA.BC(i,:)=-z-1/2*vo2+vco2; %CO + 1/2O2 -> CO2
47      CANTERA.O2_NEEDED(i) = 1/2;
48      CANTERA.ELECTRONS(i) = 2;
49      CANTERA.FUELSTREAMS(i) = 1;
50      CANTERA.ELEM(elementIndex(gas,'C'),i)=1;
51      CANTERA.ELEM(elementIndex(gas,'O'),i)=1;
52      CANTERA.HV(i) = -10102.6301773312;
53    case 'CO2',
54      CANTERA.I.CO2=i;
55      CANTERA.Hf(i)=-393.522;  %CO2 +-0.05 kJ/mol
56      CANTERA.BC(i,:)=z;       %CO2->CO2
57      CANTERA.O2_NEEDED(i) = 0;
58      CANTERA.ELECTRONS(i) = 0;
59      CANTERA.FUELSTREAMS(i) = 0;
60      CANTERA.ELEM(elementIndex(gas,'C'),i)=1;
61      CANTERA.ELEM(elementIndex(gas,'O'),i)=2;
62      CANTERA.HV(i) = 0;
63    case 'H2',
64      CANTERA.I.H2=i;
65      CANTERA.Hf(i)=0;         %H2
66      z(i)=1;
67      CANTERA.BC(i,:)=-z-1/2*vo2+vh2o; %H2 + 1/2O2 -> H2O
68      CANTERA.O2_NEEDED(i) = 1/2;
69      CANTERA.ELECTRONS(i) = 2;
70      CANTERA.FUELSTREAMS(i) = 1;
71      CANTERA.ELEM(elementIndex(gas,'H'),i)=2;
72      CANTERA.HV(i) = -119959.967295009; % kJ/kg
73    case 'H2O',
74      CANTERA.I.H2O=i;
75      CANTERA.Hf(i)=-241.826;  %H2O +-0.042 kJ/mol steam
76      %CANTERA.Hf(i)=-285.830;  %H2O +-0.042 kj/mol Water
77      CANTERA.BC(i,:)=z;       %H2O -> H2O
78      CANTERA.O2_NEEDED(i) = 0;
79      CANTERA.ELECTRONS(i) = 0;
80      CANTERA.FUELSTREAMS(i) = 0;
81      CANTERA.ELEM(elementIndex(gas,'H'),i)=2;
82      CANTERA.ELEM(elementIndex(gas,'O'),i)=1;
83      CANTERA.HV(i) = 0;
84    case 'N2',
85      CANTERA.I.N2=i;
86      CANTERA.Hf(i)=0;         %N2
87      CANTERA.BC(i,:)=z;       %N2 -> N2
```

```
 88          CANTERA.O2_NEEDED(i) = 0;
 89          CANTERA.ELECTRONS(i) = 0;
 90          CANTERA.FUELSTREAMS(i) = 0;
 91          CANTERA.ELEM(elementIndex(gas,'N'),i)=2;
 92          CANTERA.HV(i) = 0;
 93        case 'O2',
 94          CANTERA.Hf(i)=0;        %O2
 95          CANTERA.I.O2=i;
 96          CANTERA.BC(i,:)=z;      %O2 -> O2
 97          CANTERA.O2_NEEDED(i) = -1;
 98          CANTERA.ELECTRONS(i) = 4;
 99          CANTERA.FUELSTREAMS(i) = 0;
100          CANTERA.ELEM(elementIndex(gas,'O'),i)=2;
101          CANTERA.HV(i) = 0;
102        case 'AR',
103          CANTERA.I.AR=i;
104          CANTERA.Hf(i)=0;        %Ar
105          CANTERA.BC(i,:)=z;      %Ar -> Ar
106          CANTERA.O2_NEEDED(i) = 0;
107          CANTERA.ELECTRONS(i) = 0;
108          CANTERA.FUELSTREAMS(i) = 0;
109          CANTERA.ELEM(elementIndex(gas,'Ar'),i)=1;
110          CANTERA.HV(i) = 0;
111        case 'CH4',
112          CANTERA.I.CH4=i;
113          CANTERA.Hf(i)=-74.873;    %CH4 +-0.34 kJ/mol
114          z(i)=1;
115          CANTERA.BC(i,:)=-z-2*vo2+vco2+2*vh2o; %CH4 + 2O2 -> CO2 + 2H2O
116          CANTERA.O2_NEEDED(i) = 2;
117          CANTERA.ELECTRONS(i) = 8;
118          CANTERA.FUELSTREAMS(i) = 1;
119          CANTERA.ELEM(elementIndex(gas,'C'),i)=1;
120          CANTERA.ELEM(elementIndex(gas,'H'),i)=4;
121          CANTERA.HV(i) = -50026.2016331462;
122        case 'C2H6',
123          CANTERA.I.C2H6=i;
124          CANTERA.Hf(i)=-82.82;    %C2H6 +-0.3 kJ/mol
125          z(i)=1;
126          CANTERA.BC(i,:)=-z-7/2*vo2+2*vco2+3*vh2o; %C2H6 + 7/2O2 -> 2CO2 +↙
3H2O
127          CANTERA.O2_NEEDED(i) = 7/2;
128          CANTERA.ELECTRONS(i) = 14;
129          CANTERA.FUELSTREAMS(i) = 1;
130          CANTERA.ELEM(elementIndex(gas,'C'),i)=2;
131          CANTERA.ELEM(elementIndex(gas,'H'),i)=6;
132          CANTERA.HV(i) = -47511.0427611432;
133        case 'C3H8',
```

```
134         CANTERA.I.C3H8=i;
135         CANTERA.Hf(i)=-104.68;   %C3H8 +-0.5 kJ/mol
136         z(i)=1;
137         CANTERA.BC(i,:)=-z-5*vo2+3*vco2+4*vh2o; %C3H8 + 5O2 -> 3CO2 +↙
4H2O
138        CANTERA.O2_NEEDED(i) = 5;
139        CANTERA.ELECTRONS(i) = 20;
140        CANTERA.FUELSTREAMS(i) = 1;
141         CANTERA.ELEM(elementIndex(gas,'C'),i)=3;
142         CANTERA.ELEM(elementIndex(gas,'H'),i)=8;
143         CANTERA.HV(i) = -46352.2011914145;
144      case 'C',
145         CANTERA.I.C=i;
146         CANTERA.Hf(i)=716.67;    %C +-0.46 kJ/mol GAS
147         %CANTERA.Hf(i)=0;         %C Graphite
148         z(i)=1;
149         CANTERA.BC(i,:)=-z-vo2+vco2; %C + O2 -> CO2
150        CANTERA.O2_NEEDED(i) = 1;
151        CANTERA.ELECTRONS(i) = 4;
152        CANTERA.FUELSTREAMS(i) = 1;
153         CANTERA.ELEM(elementIndex(gas,'C'),i)=1;
154         CANTERA.HV(i) = -92430.6840459765;
155      case 'NO',
156         CANTERA.I.NO=i;
157         CANTERA.Hf(i)=90.291;    %NO +-0.17 kJ/mol
158         z(i)=1;
159         CANTERA.BC(i,:)=-z+1/2*vn2+1/2*vo2; %NO -> 1/2N2 + 1/2O2
160        CANTERA.O2_NEEDED(i) = 0;
161        CANTERA.ELECTRONS(i) = 0;
162        CANTERA.FUELSTREAMS(i) = 0;
163         CANTERA.ELEM(elementIndex(gas,'N'),i)=1;
164         CANTERA.ELEM(elementIndex(gas,'O'),i)=1;
165         CANTERA.HV(i) = -3041.53035795718;
166      case 'NO2',
167         CANTERA.I.NO2=i;
168         CANTERA.Hf(i)=33.10;     %NO2 +-0.8 kJ/mol
169         z(i)=1;
170         CANTERA.BC(i,:)=-z+1/2*vn2+vo2; %NO2 -> 1/2N2 + O2
171        CANTERA.O2_NEEDED(i) = 0;
172        CANTERA.ELECTRONS(i) = 0;
173        CANTERA.FUELSTREAMS(i) = 0;
174         CANTERA.ELEM(elementIndex(gas,'N'),i)=1;
175         CANTERA.ELEM(elementIndex(gas,'O'),i)=2;
176         CANTERA.HV(i) = -743.236951879324;
177   end
178 end
179
```

```
180 %% Property Calcs
181 %   %Heat Values numbers used in sfun_Properties to calc energy in stream
182 % for i=1:CANTERA.NSP
183 %    CANTERA.HV(i)=CANTERA.Hf'*CANTERA.BC(i,:)'; %kJ/mol
184 % end
185 % CANTERA.HV=CANTERA.HV./CANTERA.KG_KMOL.*1000;      %J/kg
186
```

D.3. Combustor

/Users/winstonburbank/Documents/Matlab Work/PHD Lib.../sfun_Combustor.m 1 of 4

```
 1 %  Set Temp Combustor.  t,x,u,flag,Tout, hloss, Ploss, WISE
 2 %  Tout is defined: if Tout is greater than inlet temp Xfuel is found to
 3 %  create Tout.
 4 %  hloss is amount of extra fuel needed to achieve Tout
 5 %  Ploss is the constant pressure drop across this component
 6 %  F: Fuel Flow is returned
 7
 8 function [sys,x0,str,ts] = sfun_Combustor(t,x,u,flag,Tout, hloss, Ploss, WISE)
 9 global CANTERA
10 switch flag,
11   case 0,   [sys,x0,str,ts]=mdlInitializeSizes(8,1);
12             CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
13             set(CANTERA.gas{WISE},'T',1100,'P',3*oneatm,'X','CO2:0.1, H2O:0.18, N2:↙
0.71, AR:0.01');
14             CANTERA.update{WISE}=100;
15   case 1,    sys=mdlDerivatives(t,x,u);
16   case 2,    sys=mdlUpdate(t,x,u);
17   case 3,    sys=mdlOutputs(t,x,u,flag,Tout, hloss, Ploss, WISE);
18   case {10,30,99},   sys=mdlOutputs(t,x,u,flag,Tout, hloss, Ploss, WISE);
19              x0=sys; str=0; ts=0;
20             if x0<=0, sys=[u(1);u(2);u(3)*(1-Ploss);WISE];
21             else sys=[u(1)+sys;Tout;u(3)*(1-Ploss);WISE];
22             end
23
24   case 4,    sys=mdlGetTimeOfNextVarHit(t,x,u);
25   case 9,    %sys=mdlTerminate(t,x,u,Tout, hloss, Ploss, WISE);
26             sys=mdlOutputs(t,x,u,flag,Tout, hloss, Ploss, WISE);
27    otherwise,    error(['Unhandled flag = ',num2str(flag)]);
28 end %switch
29 end %sfun
30
31 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
32 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
33 sizes = simsizes;
34 sizes.NumContStates  = 0;
35 sizes.NumDiscStates  = 0;
36 sizes.NumOutputs     = out;
37 sizes.NumInputs      = in;
38 sizes.DirFeedthrough = 1;   % u is needed for the calc of outputs=1
39 sizes.NumSampleTimes = 1;   % at least one sample time is needed
40 sys = simsizes(sizes);
41 x0  = []; % initialize the initial conditions
42 str = []; % str is always an empty matrix
43 ts  = [0 0]; % initialize the array of sample times
44 end %mdlInitializeSizes
45
46 function sys=mdlDerivatives(t,x,u)
```

```
47 sys = [];
48 end %mdlDerivatives
49
50 function sys=mdlUpdate(t,x,u)
51 sys = [];
52 end %mdlUpdate
53
54
55 function [sys]=mdlOutputs(t,x,u,flag,Tout, hloss, Ploss, WISE)
56 global CANTERA
57 %% inputs%%
58 X1=u(1); T1=u(2); P1=u(3); WISE1=u(4);  %Air
59 X2=u(5); T2=u(6); P2=u(7); WISE2=u(8);  %Fuel
60 T1=temperature(CANTERA.gas{WISE1});
61
62 P1=P1*(1-Ploss);
63 % if Tout<CANTERA.TMIN, Tout=CANTERA.TMIN; end
64
65 fuel=massFractions(CANTERA.gas{WISE2});
66 air=massFractions(CANTERA.gas{WISE1});
67 % set(CANTERA.gas{WISE},'T',Tout,'P',P1*oneatm,'Y',air,'X1);
68 % equilibrate(CANTERA.gas{WISE},'TP');
69 % T=temperature(CANTERA.gas{WISE});
70 %  There is a strange problem here:  When the stream from the fuel cell
71 %  enters, the spieces may re-combust to form the higher temperature needed,
72 %  thus not requiring any fuel even though the stream initialy entered at
73 %  below Tout.
74
75 if Tout>T1 && (fuel'*CANTERA.FUELSTREAMS)>0 %Check to make sure Temp should↙
be raised and Fuel has fuel in it
76    Hin=X1*enthalpy_mass(CANTERA.gas{WISE1});
77    hfuel=enthalpy_mass(CANTERA.gas{WISE2});
78    O2available = (-CANTERA.O2_NEEDED' * (massFractions(CANTERA.gas{WISE1}) ./↙
CANTERA.KG_KMOL))*X1; %Kmol
79    O2needed = (CANTERA.O2_NEEDED' * (massFractions(CANTERA.gas{WISE2}) ./↙
CANTERA.KG_KMOL));
80    Max_Fuel=O2available/O2needed;
81    if CANTERA.update{WISE} > Max_Fuel/3
82       CANTERA.update{WISE} = Max_Fuel/6;
83    end
84 %       options=optimset('Display','off');
85 %       F = abs(fsolve(@comb,CANTERA.update{WISE}));
86 %       F = fzero(@comb,[0,Max_Fuel]);
87       F = max(0,fzero(@comb,CANTERA.update{WISE}));
88       CANTERA.update{WISE}=F;
89 %%
90 else F=0; set(CANTERA.gas{WISE},'T',T1,'P',P1*oneatm,'Y',air);
```

```
 91 %        equilibrate(CANTERA.gas{WISE},'TP');
 92 end %if
 93 %%
 94    function z=comb(F)
 95 %  Prevents negative flows and exceeding sticometric conditions
 96 %       if F<0, F=0;
 97 %       elseif F>Max_Fuel, F=Max_Fuel;
 98 %       end
 99       F = max(0, min(F,Max_Fuel));
100       set(CANTERA.gas{WISE},'T',Tout,'P',P1*oneatm,'Y',air.*X1 + fuel.*F);
101       equilibrate(CANTERA.gas{WISE},'TP');
102       Hout=enthalpy_mass(CANTERA.gas{WISE})*(X1+F);
103       z= (Hin + F*hfuel*(1-hloss)) - Hout;  % hloss is here
104       if F==0 && z>0
105          z=0;
106       end
107 %  When the Fuel Flow is set to zero but the dH is greater than zero then
108 %  we are in this strange condition of the inlet stream re-combusting to
109 %  form the higher temperature
110    end %comb
111
112 if (flag==9 || flag==99) && (CANTERA.DP==1 || CANTERA.DP==2)
113    assignin('base',['DP_Fuelin_WISE_',num2str(WISE)],F.*massFractions(CANTERA.gas↙
{WISE2}));
114 end
115
116 if (flag==9 || flag==99) && CANTERA.DATA==1
117    X = X1+X2;
118    T = temperature(CANTERA.gas{WISE});
119    P = pressure(CANTERA.gas{WISE})./oneatm;
120    H = X.* enthalpy_mass(CANTERA.gas{WISE});
121    S = X.* enthalpy_mass(CANTERA.gas{WISE});
122
123    % XTPW_inAir, XTPW_out, H,S, 1-hloss, XTPW_Fuelin
124    Z = [X1;T1;P1;WISE1; X;T;P;WISE; H;S;1-hloss;F;T2;P2;WISE2];
125    assignin('base',['Set_Temp_Combustor_',num2str(WISE)],Z);
126
127    Z = [X1;T1;P1;WISE1;X1.*massFractions(CANTERA.gas{WISE1})];
128    assignin('base',['Set_Temp_Combustor_Air_in_',num2str(WISE)],Z);
129
130    Z = [F;T2;P2;WISE2;F.*massFractions(CANTERA.gas{WISE2})];
131    assignin('base',['Set_Temp_Combustor_Fuel_in_',num2str(WISE)],Z);
132
133    Z = [X; T; P; WISE; X.*massFractions(CANTERA.gas{WISE})];
134    assignin('base',['Set_Temp_Combustor_Prod_',num2str(WISE)],Z);
135 end
136
```

```
137 sys=[F];
138 end %mdlOutputs
139
140
141 function sys=mdlGetTimeOfNextVarHit(t,x,u)
142 sampleTime = 1;    % Example, set the next hit to be one second later.
143 sys = t + sampleTime;
144 end %mdlGetTimeOfNextVarHit
145
146 function sys=mdlTerminate(t,x,u,Tout, hloss, Ploss, WISE);
147 sys = [];
148 end %mdlTerminate
149
```

D.4. Compressor Lookup

/Users/winstonburbank/Documents/Matlab Wor.../sfun_Compressor_Lookup.m 1 of 5

```
function [sys,x0,str,ts] = sfun_Compressor_Lookup(t,x,u,flag, WISE, map, PR_DP, MachSpeed_DP, SurgeMargin_DP, Eff_DP, ShaftSpeed_X, Phi_X, PR_X, Eff_X)
global CANTERA
switch flag,
  case 0,   [sys,x0,str,ts]=mdlInitializeSizes(5,4);
            CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
            set(CANTERA.gas{WISE},'T',500,'P',4*oneatm,'X','O2:0.21, N2:0.78, AR:0.01');
  case 3,   sys=mdlOutputs(u,flag, WISE, map, PR_DP, MachSpeed_DP, SurgeMargin_DP, Eff_DP, ShaftSpeed_X, Phi_X, PR_X, Eff_X);
  case 9,   sys=mdlOutputs(u,flag, WISE, map, PR_DP, MachSpeed_DP, SurgeMargin_DP, Eff_DP, ShaftSpeed_X, Phi_X, PR_X, Eff_X);sys=[];
%       sys=mdlTerminate(u,flag, WISE, map, PR_DP, MachSpeed_DP, SurgeMargin_DP, ShaftSpeed_X, Phi_X, PR_X);
  case {10,30,99},   sys=mdlOutputs(u,flag, WISE, map, PR_DP, MachSpeed_DP, SurgeMargin_DP, Eff_DP, ShaftSpeed_X, Phi_X, PR_X, Eff_X);
            x0=sys(3); % Work
            str=sys(4); % n
            sys=[u(1);sys(1);sys(2);WISE]; %XTPW
            ts=[];
  case {1,2,4}, sys=[];
  otherwise,    error(['Unhandled flag = ',num2str(flag)]);
end %switch
end %sfun

function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = out;
sizes.NumInputs      = in;
sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs
sizes.NumSampleTimes = 1;  % at least one sample time is needed
sys = simsizes(sizes);
x0  = []; % initialize the initial conditions
str = []; % str is always an empty matrix
ts  = [0 0]; % initialize the array of sample times
end %mdlInitializeSizes

function [sys] = mdlOutputs(u,flag, WISE, map, PR_DP, MachSpeed_DP, SurgeMargin_DP, Eff_DP, ShaftSpeed_X, Phi_X, PR_X, Eff_X)
global CANTERA;
X1=u(1); T1=u(2); P1=u(3)*oneatm; WISEin=u(4); ShaftSpeed = u(5);
Phi = X1 * sqrt(T1 / CANTERA.REF.T) / (P1 / CANTERA.REF.P);

if CANTERA.DP==1, %Design Point
   PR = PR_DP;
   MachSpeed_map = MachSpeed_DP;
```

```
42 %    MachSpeed_map = ShaftSpeed/sqrt(T1);
43     ShaftSpeed_X = MachSpeed_DP / (ShaftSpeed/sqrt(T1));
44    if isfield(map, 'SPEEDS')
45     PRs_map = zeros(1,10);
46     prs = interp1(map.SPEEDS(:,1),map.SPEEDS(:,2:end),↙
MachSpeed_DP,'linear','extrap');
47     PRs_map(end-length(prs)+1:end) = prs;
48     surge = zeros(1,10);
49     surge(end-length(map.SURGE)+1:end) = map.SURGE;
50     Phi_map = roots(surge - PRs_map);  %need to solve intersection of polynomials
51     for i=1:length(Phi_map)
52        if ~isreal(Phi_map(i)), Phi_map(i)=-1; end
53     end
54     Phi_map = Phi_map(Phi_map>min(map.PHI));
55     Phi_map = Phi_map(Phi_map<max(map.PHI));
56     Phi_map = Phi_map .* (1+SurgeMargin_DP);
57     if size(Phi_map)>1, disp('Compressor Error, Phi_map >1 value'), end
58     PR_map = polyval(PRs_map,Phi_map);
59     n_map  = interp2(map.PHI,map.PR,map.EFF,Phi_map,PR_map);
60     beta_value = 666;
61
62    elseif isfield(map, 'Beta')
63     beta_value = SurgeMargin_DP;
64     if MachSpeed_DP>=min(map.speeddata(:,3)) && MachSpeed_DP<=max(map.↙
speeddata(:,3))
65        Phi_map =  polyval(interp1(map.Beta.speedvalues, map.Beta.Phi,↙
MachSpeed_DP,'spline'), SurgeMargin_DP);
66        PR_map  =  polyval(interp1(map.Beta.speedvalues, map.Beta.Pr ,↙
MachSpeed_DP,'spline'), SurgeMargin_DP);
67        n_map   =  polyval(interp1(map.Beta.speedvalues, map.Beta.Eff,↙
MachSpeed_DP,'spline'), SurgeMargin_DP);
68     else
69        Phi_map =  polyval(interp1(map.Beta.speedvalues, map.Beta.Phi,↙
MachSpeed_DP,'linear','extrap'), SurgeMargin_DP);
70        PR_map  =  polyval(interp1(map.Beta.speedvalues, map.Beta.Pr ,↙
MachSpeed_DP,'linear','extrap'), SurgeMargin_DP);
71        n_map   =  polyval(interp1(map.Beta.speedvalues, map.Beta.Eff,↙
MachSpeed_DP,'linear','extrap'), SurgeMargin_DP);
72     end
73
74    else disp('Compressor map does not have SPEEDS or SPEED field!')
75    end
76    PR_X = PR_DP / PR_map;
77    Phi_X = Phi / Phi_map;
78    Eff_X = Eff_DP / n_map;  n = Eff_DP;
79 %    Eff_X = Eff_DP;        n = n_map * Eff_X;
80
```

```
 81    if flag == 9 || flag==99
 82       disp(['For WISE # ',num2str(WISE)]);
 83       disp(['ShaftSpeed_X should equal ',num2str(ShaftSpeed_X)]);
 84       disp(['PR_X adjustment in should equal ',num2str(PR_X)]);
 85       disp(['Phi_X adjustment in should equal ',num2str(Phi_X)]);
 86       disp(['Eff_X adjustment in should equal ',num2str(Eff_X)]);
 87       disp(['Compressor Eff is ',num2str(n)]);
 88 %       disp(['Max Compressor Eff is ',num2str(max(max(map.EFF))*Eff_X)]);
 89 %       n_map2 = interp2(map.PHI, map.PR, map.EFF, Phi_map, PR_map);
 90 %       disp(['Original Compressor Eff map is ',num2str(n_map2)]);
 91       disp(' ');
 92       assignin('base',['DP_WISE_',num2str(WISE)],[ShaftSpeed_X,PR_X,Phi_X,Eff_X]);
 93    end
 94 else % CANTERA.DP==0
 95    MachSpeed_map = ShaftSpeed/sqrt(T1) * ShaftSpeed_X;
 96
 97    if isfield(map, 'SPEEDS')
 98    PR = polyval(interp1(map.SPEEDS(:,1),map.SPEEDS(:,2:end),↙
MachSpeed_map,'linear','extrap'),Phi/Phi_X)*PR_X;
 99    if PR<1, PR = 1, end
100    n = interp2(map.PHI, map.PR, map.EFF, Phi/Phi_X, PR/PR_X) * Eff_X;
101    beta_value = 666;
102
103    elseif isfield(map, 'Beta')
104    betas = 0:0.01:1;
105    if MachSpeed_map>=min(map.speeddata(:,3)) && MachSpeed_map<=max(map.↙
speeddata(:,3))
106       method = 'spline';
107    else
108       method = 'linear';
109    end
110    Phis_map =  polyval(interp1(map.Beta.speedvalues, map.Beta.Phi,↙
MachSpeed_map,method,'extrap'), betas);
111    beta_value = interp1(Phis_map, betas, Phi/Phi_X,'spline','extrap');
112 %    beta = max(0, min(1,beta));
113    if beta_value<0 || beta_value>1
114       betas = [0, 0.1, 0.9, 1]; %0:0.1:1; %More Corse to help linear extrap off the↙
ends
115       Phis_map =  polyval(interp1(map.Beta.speedvalues, map.Beta.Phi,↙
MachSpeed_map,method,'extrap'), betas);
116       beta_value = interp1(Phis_map, betas, Phi/Phi_X,'linear','extrap');
117       PRs= polyval(interp1(map.Beta.speedvalues, map.Beta.Pr, MachSpeed_map,↙
method,'extrap'), betas)*PR_X;
118       ns = polyval(interp1(map.Beta.speedvalues, map.Beta.Eff, MachSpeed_map,↙
method,'extrap'), betas) * Eff_X;
119       PR = interp1(betas, PRs, beta_value,'linear','extrap');
120       n  = interp1(betas,  ns, beta_value,'linear','extrap');
```

```
121
122    else  %beta = interp1(Phis_map, betas, Phi/Phi_X,'linear','extrap');
123      PR= polyval(interp1(map.Beta.speedvalues, map.Beta.Pr, MachSpeed_map,↙
method,'extrap'), beta_value)*PR_X;
124      n = polyval(interp1(map.Beta.speedvalues, map.Beta.Eff, MachSpeed_map,↙
method,'extrap'), beta_value) * Eff_X;
125
126    end
127   else disp('Compressor map does not have SPEEDS or Beta field!')
128   end
129    if n<=0 || n>1 || isnan(n),
130      if CANTERA.ERROR >= 1, disp(['WISE = ',num2str(WISE),' Problem with n= ',↙
num2str(n)]); end
131 %       n=map.EFF(1,1)*Eff_X,
132       n=0.666,
133    end %if
134 end
135
136 P2 = P1 * PR;
137 h1=enthalpy_mass(CANTERA.gas{WISEin});
138 set(CANTERA.gas{WISE},'S',entropy_mass(CANTERA.gas{WISEin}),'P',P2,'Y',↙
massFractions(CANTERA.gas{WISEin}));
139 h2s=enthalpy_mass(CANTERA.gas{WISE});
140 if PR>=1, work=(h2s-h1)/n; %Compressor
141 else      work=(h2s-h1)*n; if CANTERA.ERROR >= 1, disp(['WISE = ',num2str(WISE),'↙
Error in Compressor_Lookup PR <1']), end %Turbine
142 end %if
143 h2=h1+work;
144 setState_HP(CANTERA.gas{WISE},[h2,P2]);
145 T2=temperature(CANTERA.gas{WISE});
146 Work = work * X1 / 1000;  %kW
147
148 if flag == 9 || flag==99 % terminate
149    if CANTERA.PLOT==1
150       figure(WISE), hold off,
151       if isfield(map, 'SPEEDS')
152          y = polyval(interp1(map.SPEEDS(:,1),map.SPEEDS(:,2:end),↙
MachSpeed_map,'linear','extrap'),map.PHI);
153          plot(map.PHI.*Phi_X, y.*PR_X,'k-')
154          axis([map.PHI(1)*Phi_X, map.PHI(end)*Phi_X, map.PR(1)*PR_X, map.PR(end)↙
*PR_X]);
155          contour(map.PHI.*Phi_X, map.PR.*PR_X, map.EFF.*Eff_X); grid on %colorbar,
156       elseif isfield(map, 'Beta')
157          beta = 0:0.005:1;
158          Phi_beta =  polyval(interp1(map.Beta.speedvalues, map.Beta.Phi,↙
MachSpeed_map,'linear','extrap'), beta).*Phi_X;
159          PR_beta =  polyval(interp1(map.Beta.speedvalues, map.Beta.Pr ,↙
```

```
MachSpeed_map,'linear','extrap'), beta).*PR_X;
160         EFF_beta = polyval(interp1(map.Beta.speedvalues, map.Beta.Eff,↙
MachSpeed_map,'linear','extrap'), beta).*Eff_X;
161         plot(Phi_beta, PR_beta, 'k-','Linewidth',2), grid on
162       end
163       hold on
164       plot(map.speeddata(:,1).*Phi_X, map.speeddata(:,2).*PR_X,'b.',...
165          Phi,PR,'ro',...
166          map.surgedata(:,1).*Phi_X, map.surgedata(:,2).*PR_X,'g.','LineWidth',2),
167 %       hold on, contour(map.PHI.*Phi_X,map.PR.*PR_X,map.EFF.*Eff_X,↙
[0.85,0.8,0.75,0.7]), colorbar
168       title(sprintf(['eff = ',num2str(n),'    Phi = ',num2str(Phi),'    PR = ',num2str↙
(PR),...
169          '\n MachSpeed = ',num2str(MachSpeed_map),'  ShaftSpeed = ',num2str↙
(ShaftSpeed),'   Work = ',num2str(Work)]))
170       xlabel('Phi'),ylabel('PR')
171    end %if CANTERA.PLOT==1
172    if CANTERA.DATA==1
173       H = X1.* enthalpy_mass(CANTERA.gas{WISE});
174       S = X1.* entropy_mass(CANTERA.gas{WISE});
175       SurgeMargin = (interp1(map.surgedata(:,1) .* Phi_X, map.surgedata(:,2) .*↙
PR_X, Phi,'linear','extrap') ./ PR)-1;
176       Z = [X1;T1;P1/oneatm;WISEin; X1;T2;P2/oneatm;WISE; H;S;n;Work;ShaftSpeed;↙
beta_value;SurgeMargin];
177       assignin('base',['Compressor_',num2str(WISE)],Z);
178    end%if CANTERA.DATA==1
179 end %flag ==9
180
181 sys=[T2, P2/oneatm, Work, n];
182 end %mdlOutputs
183
184 % function sys = mdlTerminate(u,flag, WISE, map, PR_DP, MachSpeed_DP,↙
SurgeMargin_DP, ShaftSpeed_X, Phi_X, PR_X)
185 % sys =[];
186 % end %mdlTerminate
187
188
189
```

D.5. Equilibrium

/Users/winstonburbank/Documents/Matlab Work/PHD Lib.../sfun_Equilibrium.m 1 of 2

```
 1 %sfun_Compressor will aidabatically compress or expand a gas stream
 2
 3
 4 function [sys,x0,str,ts] = sfun_Equilibrium(t,x,u,flag,n_Fuel, ploss, hloss, WISE)
 5 global CANTERA
 6 switch flag,
 7  case 0,   [sys,x0,str,ts]=mdlInitializeSizes(8,1);
 8          CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
 9          set(CANTERA.gas{WISE},'T',1100,'P',3*oneatm,'X','CO2:0.1, H2O:0.18, N2:0.71,↙
AR:0.01');
10  case 1,   sys=mdlDerivatives(t,x,u);
11  case 2,   sys=mdlUpdate(t,x,u);
12  case 3,   sys=mdlOutputs(t,x,u,n_Fuel, ploss, hloss, WISE);
13  case 10,  sys=mdlOutputs(t,x,u,n_Fuel, ploss, hloss, WISE);   x0=0; str=0; ts=0;
14  case 4,   sys=mdlGetTimeOfNextVarHit(t,x,u);
15  case 9,   sys=mdlTerminate(t,x,u);
16   otherwise,   error(['Unhandled flag = ',num2str(flag)]);
17 end %switch
18 %end %sfun
19
20 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
21 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
22 sizes = simsizes;
23 sizes.NumContStates  = 0;
24 sizes.NumDiscStates  = 0;
25 sizes.NumOutputs     = out;
26 sizes.NumInputs      = in;
27 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs=1
28 sizes.NumSampleTimes = 1;  % at least one sample time is needed
29 sys = simsizes(sizes);
30 x0  = []; % initialize the initial conditions
31 str = []; % str is always an empty matrix
32 ts  = [0 0]; % initialize the array of sample times
33 %end %mdlInitializeSizes
34
35 function sys=mdlDerivatives(t,x,u)
36 sys = [];
37 %end %mdlDerivatives
38
39 function sys=mdlUpdate(t,x,u)
40 sys = [];
41 %end %mdlUpdate
42
43
44 function [sys]=mdlOutputs(t,x,u,n_Fuel, ploss, hloss, WISE)
45 global CANTERA
46 %%Inputs%%
```

```
47 X1=u(1); T1=u(2); P1=u(3); WISEin1=u(4);
48 X2=u(5); T2=u(6); P2=u(7); WISEin2=u(8);
49
50 % Need to add n_Fuel
51 Mf1= X1*massFractions(CANTERA.gas{WISEin1});
52 H1 = X1*enthalpy_mass(CANTERA.gas{WISEin1});
53
54 Mf2= X2*massFractions(CANTERA.gas{WISEin2});
55 H2 = X2*enthalpy_mass(CANTERA.gas{WISEin2});
56
57 X3=X1+X2;
58 P3=min(P1,P2)*(1-ploss); %ploss (atm)
59 if X3==0, set(CANTERA.gas{WISE},'T',temperature(CANTERA.gas{WISEin1}),'P',↙
pressure(CANTERA.gas{WISEin1}),'Y',massFractions(CANTERA.gas{WISEin1}));
60 else set(CANTERA.gas{WISE},'H',(H1+H2)/X3,'P',P3*oneatm,'Y',(Mf1+Mf2)./X3);
61 end
62 % Warning problems seem to occur when products are at low temperatures or
63 % way too much fuel for air
64 equilibrate(CANTERA.gas{WISE},'HP');
65 T3=temperature(CANTERA.gas{WISE});
66 % Need to add hloss
67
68 sys=[T3];
69 %end %mdlOutputs
70
71
72 function sys=mdlGetTimeOfNextVarHit(t,x,u)
73 sampleTime = 1;    % Example, set the next hit to be one second later.
74 sys = t + sampleTime;
75 %end %mdlGetTimeOfNextVarHit
76
77 function sys=mdlTerminate(t,x,u)
78 sys = [];
79 %end %mdlTerminate
80
```

D.6. Equilibrium Basic

/Users/winstonburbank/Documents/Matlab Work/P.../sfun_Equilibrium_Basic.m 1 of 4

```
 1
 2
 3 function [sys,x0,str,ts] = sfun_Equilibrium_Basic(t,x,u,flag,n_Fuel, hloss, Ploss, WISE)
 4 global CANTERA
 5 switch flag,
 6  case 0,  [sys,x0,str,ts]=mdlInitializeSizes(8,1);
 7          CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
 8          set(CANTERA.gas{WISE},'T',1100,'P',3*oneatm,'X','CO2:0.1, H2O:0.18, N2:0.71,↙
AR:0.01');
 9  case 1,   sys=mdlDerivatives(t,x,u);
10  case 2,   sys=mdlUpdate(t,x,u);
11  case 3,   sys=mdlOutputs(t,x,u, hloss, Ploss, WISE);
12  case 10,  sys=mdlOutputs(t,x,u, hloss, Ploss, WISE);   x0=0; str=0; ts=0;
13  case 4,   sys=mdlGetTimeOfNextVarHit(t,x,u);
14  case 9,   sys=mdlTerminate(t,x,u, hloss, Ploss, WISE);
15   otherwise,   error(['Unhandled flag = ',num2str(flag)]);
16 end %switch
17 end %sfun
18
19 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
20 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
21 sizes = simsizes;
22 sizes.NumContStates  = 0;
23 sizes.NumDiscStates  = 0;
24 sizes.NumOutputs     = out;
25 sizes.NumInputs      = in;
26 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs=1
27 sizes.NumSampleTimes = 1;  % at least one sample time is needed
28 sys = simsizes(sizes);
29 x0  = []; % initialize the initial conditions
30 str = []; % str is always an empty matrix
31 ts  = [0 0]; % initialize the array of sample times
32 end %mdlInitializeSizes
33
34 function sys=mdlDerivatives(t,x,u)
35 sys = [];
36 end %mdlDerivatives
37
38 function sys=mdlUpdate(t,x,u)
39 sys = [];
40 end %mdlUpdate
41
42
43 function [sys]=mdlOutputs(t,x,u, hloss, Ploss, WISE)
44 global CANTERA
45 %% inputs%%
46 X1=u(1); T1=u(2); P1=u(3); WISE1=u(4);  %Air
```

```
47 X2=u(5); T2=u(6); P2=u(7); WISE2=u(8);  %Fuel
48 T1=temperature(CANTERA.gas{WISE1});
49
50 % P3 = P1 * (1-Ploss);
51 P3=min(P1,P2)*(1-Ploss);
52
53 h1 = (X1*enthalpy_mass(CANTERA.gas{WISE1}) + X2*enthalpy_mass(CANTERA.gas{WISE2})) / (X1+X2);
54 Mix = X1.*massFractions(CANTERA.gas{WISE1}) + X2.*massFractions(CANTERA.gas{WISE2});
55 molMix = Mix ./ CANTERA.KG_KMOL;
56 atomMix = CANTERA.ELEM * molMix;  % O H C N Ar
57 molProd = zeros(size(molMix));
58
59 %% C + O -> CO
60     X = min(atomMix(1),atomMix(3));
61     atomMix(1) = atomMix(1) - X;
62     atomMix(3) = atomMix(3) - X;
63     molProd(CANTERA.I.CO) = molProd(CANTERA.I.CO) + X;
64
65 %% CO + 2H + 2O -> CO2 + H2O
66     X = min([atomMix(1)/2, atomMix(2)/2, molProd(CANTERA.I.CO)]);
67     atomMix(1) = atomMix(1) - 2*X;
68     atomMix(2) = atomMix(2) - 2*X;
69     molProd(CANTERA.I.CO) = molProd(CANTERA.I.CO) - X;
70     molProd(CANTERA.I.CO2) = molProd(CANTERA.I.CO2) + X;
71     molProd(CANTERA.I.H2O) = molProd(CANTERA.I.H2O) + X;
72
73 %% CO + O -> CO2
74     X = min(atomMix(1), molProd(CANTERA.I.CO));
75     atomMix(1) = atomMix(1) - X;
76     molProd(CANTERA.I.CO) = molProd(CANTERA.I.CO) - X;
77     molProd(CANTERA.I.CO2) = molProd(CANTERA.I.CO2) + X;
78
79 %% 2H + O -> H2O
80     X = min(atomMix(1), atomMix(2)/2);
81     atomMix(1) = atomMix(1) - X;
82     atomMix(2) = atomMix(2) - 2*X;
83     molProd(CANTERA.I.H2O) = molProd(CANTERA.I.H2O) + X;
84
85 %% C + 4H -> CH4
86     X = min(atomMix(3), atomMix(2)/4);
87     atomMix(3) = atomMix(3) - X;
88     atomMix(2) = atomMix(2) - 4*X;
89     molProd(CANTERA.I.CH4) = molProd(CANTERA.I.CH4) + X;
90
91 %% 2H -> H2
```

```
 92    molProd(CANTERA.I.H2) = atomMix(2)/2; % + molProd(CANTERA.I.H2)
 93 %     atomMix(2) = 0;
 94
 95 %% 2O -> O2
 96    molProd(CANTERA.I.O2) = atomMix(1)/2 ; % + molProd(CANTERA.I.O2)
 97 %     atomMix(1) = 0;
 98
 99 %% 2N --> N2
100    molProd(CANTERA.I.N2) = atomMix(4)/2 ; % + molProd(CANTERA.I.N2)
101 %     atomMix(4) = 0;
102
103 %% Ar
104    molProd(CANTERA.I.AR) = atomMix(5) ; % + molProd(CANTERA.I.AR)
105 %     atomMix(5) = 0;
106 %
107 %     atomMix2 = CANTERA.ELEM * molProd;  % O H C N Ar
108
109 set(CANTERA.gas{WISE},'H',h1,'P',P3 *oneatm,'X', molProd);
110 T3 = temperature(CANTERA.gas{WISE});
111 sys=[T3];
112 end %mdlOutputs
113
114
115 function sys=mdlGetTimeOfNextVarHit(t,x,u)
116 sampleTime = 1;    %  Example, set the next hit to be one second later.
117 sys = t + sampleTime;
118 end %mdlGetTimeOfNextVarHit
119
120 function sys=mdlTerminate(t,x,u, hloss, Ploss, WISE);
121 global CANTERA
122 if CANTERA.DATA==1
123    X1=u(1); T1=u(2); P1=u(3); WISE1=u(4);  %Air
124    X2=u(5); T2=u(6); P2=u(7); WISE2=u(8);  %Fuel
125
126    X = X1+X2;
127    T = temperature(CANTERA.gas{WISE});
128    P = pressure(CANTERA.gas{WISE})./oneatm;
129    H = X.* enthalpy_mass(CANTERA.gas{WISE});
130    S = X.* enthalpy_mass(CANTERA.gas{WISE});
131
132    % XTPW_in, XTPW_out, HS, 1-hloss, 0000
133    Z = [X1;X2;WISE1;WISE2; X;T;P;WISE; H;S;1-hloss;0;0;0;0];
134    assignin('base',['Set_Temp_Combustor_',num2str(WISE)],Z);
135
136    Z = [X1;T1;P1;WISE1;X1.*massFractions(CANTERA.gas{WISE1})];
137    assignin('base',['Set_Temp_Combustor_Air_in_',num2str(WISE)],Z);
138
```

```
139     Z = [X2;T2;P2;WISE2;X2.*massFractions(CANTERA.gas{WISE2})];
140     assignin('base',['Set_Temp_Combustor_Fuel_In_',num2str(WISE)],Z);
141
142     Z = [X; T; P; WISE; X.*massFractions(CANTERA.gas{WISE})];
143     assignin('base',['Set_Temp_Combustor_Prod_',num2str(WISE)],Z);
144 end%if
145 sys = [];
146 end %mdlTerminate
147
```

D.7. Fuel Cell

/Users/winstonburbank/Documents/Matlab Work/PHD Li.../sfun_Fuel_Cell.m 1 of 16

```
 1 %sfun_Compressor will aidabatically compress or expand a gas stream
 2
 3
 4 function [sys,x0,str,ts] = sfun_Fuel_Cell(t,x,u,flag,...
 5    FC_Type, n_Fuel, rec, Ploss_anode, Ploss_cathode, Ploss_combustor, Ploss_HX,↙
FC_Volt, HX_Eff, CellArea, CellTemp, FCExitTemp,...
 6    WISEAirHX, WISEAirFC, WISEPComb, WISEPHX, WISEFuelReform, WISEFuelFC,↙
WISErec1, WISErec2)
 7 global CANTERA
 8 switch flag,
 9  case 0,   [sys,x0,str,ts]=mdlInitializeSizes(8,36);
10          CANTERA.gas{WISEAirHX} = importPhase('Basic.cti',CANTERA.GAS);
11          set(CANTERA.gas{WISEAirHX},'T',1100,'P',3*oneatm,'X','N2:0.78, O2:0.21, AR:↙
0.01');
12
13          CANTERA.gas{WISEAirFC} = importPhase('Basic.cti',CANTERA.GAS);
14          set(CANTERA.gas{WISEAirFC},'T',1273,'P',3*oneatm,'X','N2:0.83, O2:0.16, AR:↙
0.01');
15
16          CANTERA.gas{WISEPComb} = importPhase('Basic.cti',CANTERA.GAS);
17          set(CANTERA.gas{WISEPComb},'T',1300,'P',3*oneatm,'X','CO2:0.033, H2O:↙
0.067, N2:0.73, O2:0.17');
18
19          CANTERA.gas{WISEPHX} = importPhase('Basic.cti',CANTERA.GAS);
20          set(CANTERA.gas{WISEPHX},'T',1100,'P',3*oneatm,'X','CO2:0.033, H2O:0.067,↙
N2:0.73, O2:0.17');
21
22          CANTERA.gas{WISEFuelReform} = importPhase('Basic.cti',CANTERA.GAS);
23          set(CANTERA.gas{WISEFuelReform},'T',1273,'P',3*oneatm,'X','CO2:0.50, H2O:↙
0.15, CO:0.20, H2:0.15');
24
25          CANTERA.gas{WISEFuelFC} = importPhase('Basic.cti',CANTERA.GAS);
26          set(CANTERA.gas{WISEFuelFC},'T',1273,'P',3*oneatm,'X','CO2:0.33, H2O:↙
0.67');
27
28          CANTERA.gas{WISErec1} = importPhase('Basic.cti',CANTERA.GAS);
29          set(CANTERA.gas{WISErec1},'T',1100,'P',3*oneatm,'X','CO2:0.50, H2O:0.15,↙
CO:0.20, H2:0.15');
30
31          CANTERA.gas{WISErec2} = importPhase('Basic.cti',CANTERA.GAS);
32          set(CANTERA.gas{WISErec2},'T',1100,'P',3*oneatm,'X','CO2:0.50, H2O:0.15,↙
CO:0.20, H2:0.15');
33
34          CANTERA.update{45}=666;
35 %          CANTERA.update{u(4)}=666;
36
37  case 1,   sys=mdlDerivatives();
```

```
 38  case 2,   sys=mdlUpdate();
 39  case 3,   sys=mdlOutputs(t,x,u,flag, FC_Type, n_Fuel, rec, Ploss_anode,↙
Ploss_cathode, Ploss_combustor, Ploss_HX,  FC_Volt, HX_Eff, CellArea, CellTemp,↙
FCExitTemp, WISEAirHX, WISEAirFC, WISEPComb, WISEPHX, WISEFuelReform, WISEFuelFC,↙
WISErec1, WISErec2);
 40  case {10,30,99},  sys=mdlOutputs(t,x,u,flag, FC_Type, n_Fuel, rec, Ploss_anode,↙
Ploss_cathode, Ploss_combustor, Ploss_HX,  FC_Volt, HX_Eff, CellArea, CellTemp,↙
FCExitTemp, WISEAirHX, WISEAirFC, WISEPComb, WISEPHX, WISEFuelReform, WISEFuelFC,↙
WISErec1, WISErec2);
 41            x0=0; str=0; ts=0;
 42  case 4,   sys=t+1;
 43  case 9,   sys=mdlOutputs(t,x,u,flag, FC_Type, n_Fuel, rec, Ploss_anode,↙
Ploss_cathode, Ploss_combustor, Ploss_HX,  FC_Volt, HX_Eff, CellArea, CellTemp,↙
FCExitTemp, WISEAirHX, WISEAirFC, WISEPComb, WISEPHX, WISEFuelReform, WISEFuelFC,↙
WISErec1, WISErec2);
 44            sys=[];
 45  otherwise,   error(['Unhandled flag = ',num2str(flag)]);
 46 end %switch
 47 end %sfun
 48
 49 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
 50 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
 51 sizes = simsizes;
 52 sizes.NumContStates  = 0;
 53 sizes.NumDiscStates  = 0;
 54 sizes.NumOutputs     = out;
 55 sizes.NumInputs      = in;
 56 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs=1
 57 sizes.NumSampleTimes = 1;  % at least one sample time is needed
 58 sys = simsizes(sizes);
 59 x0  = []; % initialize the initial conditions
 60 str = []; % str is always an empty matrix
 61 ts  = [0 0]; % initialize the array of sample times
 62 end %mdlInitializeSizes
 63
 64 function sys=mdlDerivatives()
 65 sys = [];
 66 end %mdlDerivatives
 67
 68 function sys=mdlUpdate()
 69 sys = [];
 70 end %mdlUpdate
 71
 72
 73 function [sys]=mdlOutputs(t,x,u,flag, FC_Type, n_Fuel, rec, Ploss_anode,↙
Ploss_cathode, Ploss_combustor, Ploss_HX,  FC_Volt, HX_Eff, CellArea, CellTemp,↙
FCExitTemp, WISEAirHX, WISEAirFC, WISEPComb, WISEPHX, WISEFuelReform, WISEFuelFC,↙
```

```
WISErec1, WISErec2)
 74 %%
 75    global CANTERA
 76    %%Inputs%%
 77    xFuelin = 1; %Assumption of unity
 78 %    XFuelin = u(1);
 79    TFuelin=u(2); PFuelin=u(3); WISEFuelin=u(4);  %Fuel
 80    XAirin=u(5); TAirin=u(6); PAirin=u(7); WISEAirin=u(8);  %Air
 81
 82    %  Air heat exchanger
 83    XAirHX = XAirin;
 84
 85    %  Fuel / Anode Inlet
 86    hFuelin = xFuelin * enthalpy_mass(CANTERA.gas{WISEFuelin});  %W/1(kg/s)
 87    mfFuelin= xFuelin .* massFractions(CANTERA.gas{WISEFuelin});  %(kg/s)/1(kg/s)
 88    molsFuelin = mfFuelin .* CANTERA.KMOL_KG;  %(kMols/s)/1(kg/s)
 89
 90    %  Air / Cathode Inlet
 91    HAirin = XAirin * enthalpy_mass(CANTERA.gas{WISEAirin});  %W
 92    MfAirin= XAirin * massFractions(CANTERA.gas{WISEAirin});  %kg/s
 93    MolsAirin = MfAirin .* CANTERA.KMOL_KG;  %kMols/s
 94
 95    % Pressure Losses
 96    PAirHX = PAirin;
 97    PAirFC = PAirHX * (1-Ploss_cathode);
 98    PPComb = PAirFC * (1-Ploss_combustor);
 99    PPHX   = PPComb * (1-Ploss_HX);
100     PFuelReform = PFuelin * 1; % Pressure loss through injector
101     PFuelFC = PFuelReform * (1-Ploss_anode);
102
103     % Set known temperatures
104     TFuelFC = CellTemp;
105     TAirFC  = CellTemp;
106
107     % Predict Fuel Cell Molar Concentrations
108     % Assume enough O2
109     % Because n_Fuel is set and we assume there will always be enough O2
110     % We can find the fuel side composition and temperature without knowing
111     % the true fuel flow
112
113 %%  Types of Fuel Cells
114 %   1: SOFC
115     molsAcross = zeros(CANTERA.NSP,1); molsAcrossNeeded = molsAcross;↙
MolsAcrossAvailable = molsAcross;
116
117     if FC_Type==1  %SOFC
118         MolsAcrossAvailable(CANTERA.I.O2) = MolsAirin(CANTERA.I.O2);
```

```
119       molsAcrossNeeded(CANTERA.I.O2) = molsFuelin' * CANTERA.O2_NEEDED;
120       MaxFuel = min(MolsAcrossAvailable ./ molsAcrossNeeded);
121    % Assume there will always be enough O2 and Calculate FuelFC Flows
122       molsAcross = molsAcrossNeeded .* n_Fuel;
123       mfAcross = molsAcross .* CANTERA.KG_KMOL;
124       xAcross = sum(mfAcross);
125       molsFuelFC = molsFuelin + molsAcross;
126       mfFuelFC = molsFuelFC .* CANTERA.KG_KMOL;
127       xFuelFC = xFuelin + xAcross;
128 %% SOFC Combustion at the exit of Fuel Cell Stack
129       set(CANTERA.gas{WISEFuelFC},'T',TFuelFC,'P',PFuelFC*oneatm,'X',molsFuelFC);
130       equilibrate(CANTERA.gas{WISEFuelFC},'TP'); % Gibbs Minimum
131 %       equilibrate_basic(WISEFuelFC,'TP');  % Basic Combustion
132    % Double Check conservation of Elements
133 %       mfFuelFC2= massFractions(CANTERA.gas{WISEFuelFC}),
134 %       atomFC2= CANTERA.ELEM * moleFractions(CANTERA.gas{WISEFuelFC}),
135       mfFuelFC = xFuelFC .* massFractions(CANTERA.gas{WISEFuelFC});
136       molsFuelFC = mfFuelFC .* CANTERA.KMOL_KG;
137       hFuelFC = enthalpy_mass(CANTERA.gas{WISEFuelFC}); % Without an X↙
(assuming 1), it is used in solving loop
138
139    % Set recycle states
140       Trec1 = CellTemp;
141       Trec2 = CellTemp;
142       Prec1 = PFuelFC;
143       Prec2 = PFuelin *(1+0.05); % 5% more pressure
144       set(CANTERA.gas{WISErec1},'T',Trec1,'P',Prec1*oneatm,'X',molsFuelFC);
145       set(CANTERA.gas{WISErec2},'T',Trec2,'P',Prec2*oneatm,'X',molsFuelFC);
146
147       xrec1 = xFuelFC * rec/(1-rec); % Flow based on FC boundry layer
148       mfrec1 = mfFuelFC .* (rec / (1-rec));
149       hrec1 = xrec1 * enthalpy_mass(CANTERA.gas{WISErec1});
150    % Rec 2 is unused/not needed in SOFC
151
152    % Set Fuel Reform State
153       mfFuelReform = mfFuelin + mfrec1;
154       molsFuelReform = mfFuelReform .* CANTERA.KMOL_KG;
155       hFuelReform = hFuelin + hrec1;
156       xFuelReform = xFuelin + xrec1;
157 %       mfFuelReform = mfFuelFC  / (1-rec) - mfAcross; % Double Check
158 %       xFuelReform2 = xFuelFC / (1-rec) - xAcross;
159 %% SOFC Combustion at Fuel Reform begining of stack enterance
160       set(CANTERA.gas{WISEFuelReform},'H',hFuelReform / xFuelReform,'P',↙
PFuelReform*oneatm,'X',molsFuelReform);
161 %       equilibrate(CANTERA.gas{WISEFuelRefcrm},'HP'); % Gibbs Minimume
162       equilibrate_basic(WISEFuelReform,'HP');  % Basic Combustion
163       TFuelReform = temperature(CANTERA.gas{WISEFuelReform});
```

```
164     % Unneeded in rest of loop
165 %       mfFuelReform = xFuelReform .* massFractions(CANTERA.gas↙
{WISEFuelReform});
166 %       molsFuelReform = mfFuelReform ./ CANTERA.KG_KMOL;
167 %% SOFC VI curve: V(mV)= (dVJ + dVT) *J(mA/cm^2) + Vo + dVP + dVoxygen +↙
dVfuel
168 %       dVJ = -0.6196;  %From FC Handbook 7-22: P=5atm  mV*cm^2/mA
169 %       Vo  = 910.76; %From FC Handbook 7-22: P=5atm  mV
170       dVP   = 59 * log10(PAirFC); % (mV) From FC Handbook 7-22
171       dVJ_J = 0;
172  %  dVT_J and Vo where taken from 7-14 and adjusted to fit 1 atm on 7-14
173       dVT_J = exp(-0.00857925266551639  .* (TAirFC + -1067.17985119132)) .*↙
-0.112251785975856 + -0.535595739677627; % mV/(mA/cm^2)
174       Vo    = -0.276216130492928 .* TAirFC + 1164.82044253251; % mV
175    %    dVT = 0; % Since cell is held at 1000C  mV*cm^2/mA
176       % From FC Handbook 7-11 through 7-13
177 %       if TAirFC >= 900+273.15, dVT = 0.008 * (TAirFC-1273.15);
178 %       elseif TAirFC >= 800+273.15, dVT = 0.04 * (TAirFC-1273.15);
179 %       elseif TAirFC >= 650+273.15, dVT = 1.3 * (TAirFC-1273.15);
180 %       else dVT = 1.3 * (TAirFC-1273.15); disp('TAirFC is below 650C!!');
181 %       end %if
182       molFracO2in = moleFraction(CANTERA.gas{WISEAirin},'O2');
183       % dVoxygen set in solver loop based upon O2 utilization
184       molFracFuelReform = moleFractions(CANTERA.gas{WISEFuelReform});
185       molFracFuelFC = moleFractions(CANTERA.gas{WISEFuelFC});
186       avgmolFracH2  = (molFracFuelFC(CANTERA.I.H2) + molFracFuelReform↙
(CANTERA.I.H2))/2;
187       avgmolFracH2O = (molFracFuelFC(CANTERA.I.H2O) + molFracFuelReform↙
(CANTERA.I.H2O))/2;
188       dVfuel = 172 * log10((avgmolFracH2/avgmolFracH2O) / (1.06212833)); %mV↙
FC Handbook 7-27
189    %    dVfuel2 = 172 * log10((avgmolFracH2/avgmolFracH2O) / (0.957721467)); %↙
mV FC Handbook 7-27
190 %       dVfuel3 = 172 * log10((avgmolFracH2/avgmolFracH2O) /↙
(0.97593413552881 6)),
191 %       dVfuel4 = 172 * log10((avgmolFracH2/avgmolFracH2O) /↙
(0.958441350160598)),
192 %       dVfuel,
193    end %FC_Type
194 %%
195    options = optimset('Display','off','TolFun',10^-8,'TolX',10^-6);  % Turn off Display
196 %    if CANTERA.update{WISEFuelin}==666
197 %       Fuelg = MaxFuel/10;
198 %    else
199 %       Fuelg = CANTERA.update{WISEFuelin};
200 %    end
201 %    Fuelg = min( MaxFuel, Fuelg);
```

```
202    if CANTERA.DP == 1 || CANTERA.DP==2,
203       if TAirin >= CellTemp, Error('TAirin > CellTemp at design point!↙
sfun_Fuel_Cell 194'), end
204
205       % Solve for XFuel to satisfy FC Exit Temp
206       XFuel = fzero(@solve_FC_fuel,[1E-8,MaxFuel],options);
207 %        XFuel = abs(fsolve(@solve_FC_fuel,MaxFuel/10,options));
208 %        CANTERA.update{WISEFuelin}=XFuel;
209       Hadditional = HFC - (HAirin + XFuel * hFuelin + workFC);
210       MfPFC;
211       TPComb = temperature(CANTERA.gas{WISEPComb});
212       TPHX = temperature(CANTERA.gas{WISEPHX});
213       %Solve what HX_Eff is needed to transfer Hadd
214       HX_Eff = fsolve(@solve_FC_HX_Eff,HX_Eff,options);
215 %        set(CANTERA.gas{WISEAirFC},'T',TAirFC,'P',PAirFC *onealm,'Y',MfAirFC); %↙
Already Set in solve_FC_fuel
216       if flag==9 || flag==99,
217        JFC = (FC_Volt * 1000 - (Vo + dVP + dVoxygen + dVfuel)) / (dVT_J); %↙
mA/cm^2
218        CellArea = IFC *1000 / JFC; %cm^2
219        disp(['For Fuel Cell WISE # ',num2str(WISEPHX)]);
220        disp(['HX_Eff should equal! ',num2str(HX_Eff)]);
221        disp(['CellArea should equal ',num2str(CellArea)]);
222        disp(['Fuel Cell Work (kW) equals ',num2str(workFC/1000)]);
223        disp(['Fuel Cell J (mA/cm^2) equals ',num2str(JFC)]);
224        disp(' ');
225       assignin('base',['DP_WISE_',num2str(WISEPHX)],[HX_Eff,CellArea, workFC/1000,↙
JFC]);
226       assignin('base',['DP_Fuelin_WISE_',num2str(WISEPHX)], XFuel.*massFractions↙
(CANTERA.gas{WISEFuelin}));
227       end
228
229    else
230       z0 = solve_FC_fuel(0);  % Run FC without Fuel
231       if z0>=0, XFuel = 0;  disp(['FC is hot enough already!  CellTemp = ',num2str↙
(CellTemp),'  and TAirin = ',num2str(TAirin)]),
232       else zmax = solve_FC_fuel(MaxFuel);  % Run FC with Max Fuel
233          if zmax<=0, disp(['Need to adjust n_Fuel from ',num2str(n_Fuel)]);
234             XFuel = MaxFuel;  Xg = MaxFuel;
235             options = optimset('Display','off','TolFun',10^-8,'TolX',10^-↙
6,'DiffMaxChange',0.01);  % Turn off Display
236 %             n_Fuel = fsolve(@solve_FC_n_Fuel,n_Fuel,options);
237             n_Fuel = fzero(@solve_FC_n_Fuel,[0.01,n_Fuel],options);
238             disp(['New n_Fuel = ',num2str(n_Fuel)]);
239          else
240 %             disp('FC Normal Operation');
241 %             disp(['z0 = ',num2str(z0) '   zmax = ',num2str(zmax),'   Max Fuel = ',↙
```

```
num2str(MaxFuel)]);
242             Xg = MaxFuel * z0/(z0-zmax);
243         %Solve for XFuel to satisfy CellTemp
244             options = optimset('Display','off','TolFun',10^-11,'TolX',10^-↙
10,'DiffMaxChange',MaxFuel*0.001);  % Turn off Display
245 %           XFuel = fzero(@solve_FC_fuel,[0,MaxFuel],options)
246 %           XAirFC=0; XFuelFC=0; XPComb=0;
247             XFuel = (fsolve(@solve_FC_fuel,Xg,options));
248 %           XFuel,
249             z = solve_FC_fuel(XFuel);
250 %           disp('After Solver');  dX2 = XAirin + XFuel - XAirFC - XFuelFC;
251 %           disp(['XAirFC=',num2str(XAirFC),'  XFuelFC=',num2str(XFuelFC),'↙
XFuel=',num2str(XFuel),'  dX=',num2str(dX2)]);
252 %           XFuel.
253 %           XFuel = abs(fsolve(@solve_FC_fuel,CANTERA.update{WISEFuelin},↙
options));
254 %           solve_FC_fuel(XFuel),
255 %           if abs(XAirin + Xg - XAirFC - XFuelFC)>1E-10
256 %              dX1 = XAirin + Xg - XAirFC - XFuelFC;
257 %              dX2 = XAirin + XFuel - XAirFC - XFuelFC;
258 %           end
259 %           dXFC = XAirin + XFuel - XAirFC - XFuelFC;
260
261         end
262       end
263 %       disp(['XFuel = ',num2str(XFuel)]);
264 %       CANTERA.update{WISEFuelin}=XFuel;
265 %       XPHX = XAirin + XFuel;
266       XPHX = XPComb;
267 %       MfPHX = XPHX .* massFractions(CANTERA.gas{WISEPComb});
268 %       MfPHX2= XPHX .* massFractions(CANTERA.gas{WISEPHX});
269       HPHX = HAirin + XFuel * hFuelin + workFC;
270       set(CANTERA.gas{WISEPHX},'H',HPHX/XPHX,'P',PPHX * oneatm);
271 %       set(CANTERA.gas{WISEPHX},'H',HPHX/XPHX,'P',PPHX * oneatm, 'Y',MfPHX);
272       TPHX = temperature(CANTERA.gas{WISEPHX});
273    end %if CANTERA.DP==1
274
275    if FC_Type == 1 %SOFC
276       Xrec1 = XFuelFC * rec / (1-rec);
277       Xrec2 = Xrec1;
278       XFuelReform = XFuel + Xrec2;
279 %       XFuelReform2 = XFuelFC / (1-rec) - XAcross;
280    end
281
282    %  Need to add hloss
283    WorkFC = workFC/1000;  %kW
284    Voltage = VFC; %V
```

```
285 %    avgmolFracO2 = (molFracO2in + moleFraction(CANTERA.gas{WISEAirFC},'O2'))/
/2;
286 %    XAirin; MfAirin; XAirFC; MfAirFC; XAcross; MfAcross;
287 %    MfAirFC(CANTERA.I.O2);
288 %    dXFC = XAirin + XFuel - XAirFC - XFuelFC;
289 %    dXSys = XAirin + XFuel - XPComb;
290 %    dXComb = XAirFC + XFuelFC - XPComb;
291 %    XFuel, XAcross,
292 %    JFC, MolsAcross,
293 %    mfFuelin, XFuel,
294 %    IFC.
295
296 %    molFracO2in,% = moleFraction(CANTERA.gas{WISEAirin},'O2');
297 %    MolFracO2out,%= moleFraction(CANTERA.gas{WISEAirFC},'O2');
298 %    avgmolFracO2,% = (MolFracO2in + MolFracO2out) /2;
299 %    dVoxygen,% = 92 * log10( avgmolFracO2 / 0.18811466754 );
300
301
302 %    disp(['JFC = ',num2str(JFC),' mA/cm^2 ; FC_Voltage = ',num2str(VFC),' V ;'
FC_Work = ',num2str(WorkFC),' kW']);
303 %    disp(['HX dH = ',num2str(dH)]);
304 %% solve_FC_n_Fuel Function
305    function z=solve_FC_n_Fuel(n_Fuelg)
306       n_Fuel = n_Fuelg;
307       FC_Fuel_Calcs();
308       z=solve_FC_fuel(Xg);
309
310    end %function solve_FC_n_Fuel
311 %% solve_FC_fuel function
312    function z=solve_FC_fuel(Xg)
313 %       Xg=min(abs(Xg),MaxFuel);
314    %Given fuelX determine FC Xflows
315    %Find Across
316       MolsAcross = molsAcross .* Xg;
317       MfAcross = mfAcross .* Xg;
318       XAcross = xAcross * Xg;
319
320       MfFuelFC = mfFuelFC .* Xg;
321       XFuelFC = xFuelFC * Xg;
322
323       MfAirFC = MfAirin - MfAcross;
324       XAirFC = XAirin - XAcross;
325
326
327    %Set Air side to CellTemp and P
328      set(CANTERA.gas{WISEAirFC},'T',TAirFC,'P',PAirFC*oneatm,'Y',MfAirFC);
329    %Save H from mixture of FC exit
```

```
330       HFC = XAirFC * enthalpy_mass(CANTERA.gas{WISEAirFC}) + XFuelFC * hFuelFC;
331     %Find dVoxygen
332       avgmolFracO2 = (molFracO2in + moleFraction(CANTERA.gas{WISEAirFC},'O2'))↙
/2;
333       dVoxygen = 92 * log10( avgmolFracO2 / 0.18811466754 );
334     %Find FC Work
335       IFC = MolsAcross' * CANTERA.ELECTRONS * 96485 * 1000;  %Amps
336       JFC = IFC * 1000 / CellArea; %mA/cm^2
337     %Combust Products after set CellTemp
338       MfPFC = MfAirFC + MfFuelFC;
339       XPComb = XAirin + Xg;
340       HPComb = HFC;
341      set(CANTERA.gas{WISEPComb},'H',HPComb/XPComb,'P',PPComb*oneatm,'Y',↙
MfPFC);
342       equilibrate(CANTERA.gas{WISEPComb},'HP');
343       TPComb = temperature(CANTERA.gas{WISEPComb});
344
345       if CANTERA.DP==1 || CANTERA.DP==2  % Solving for Discharge Temp
346          VFC = FC_Volt; %V
347          workFC = -VFC * IFC; %Watts
348       %Set Discharge state
349          XPHX = XPComb;
350          MfPHX = XPHX .* massFractions(CANTERA.gas{WISEPComb});
351          set(CANTERA.gas{WISEPHX},'T',FCExitTemp,'P',PPHX*oneatm,'Y',MfPHX);
352          HPHX = XPHX * enthalpy_mass(CANTERA.gas{WISEPHX});
353       %Compute Enthalpy Balance
354          z = Xg * hFuelin + HAirin + workFC - HPHX;
355
356       else  % Solving for CellTemp
357          VFC = ( (dVT_J) * JFC + Vo + dVP + dVoxygen + dVfuel) /1000; %V
358          workFC = -VFC * IFC; %Watts
359       %Save Hadditional: how much H was added to set CellTemp
360          Hadditional = HFC - (HAirin + Xg * hFuelin + workFC);
361       %Run through HX with known effectiveness
362          u = [XPComb; TPComb; PPComb; WISEPComb;  XAirin; TAirin; PAirin;↙
WISEAirin];
363          %[sys,x0,str,ts] = sfun_HeatExchanger(t,x,u,flag,n_HX,  ploss_hot,ploss_cold,↙
WISE_hot,WISE_cold,TCc);
364          [sys,x0,str,ts] = sfun_HeatExchanger(t,x,u, 30 ,HX_Eff,Ploss_HX, 0       ,↙
WISEPHX ,WISEAirHX,1000);
365          TPHX = sys(2); TAirHX = x0(2);
366       %Is dH across equal Hadditional?
367          dH = XAirin * enthalpy_mass(CANTERA.gas{WISEAirHX}) - HAirin;
368 %          dH2= XPComb ' (enthalpy_mass(CANTERA.gas{WISEPComb}) -↙
enthalpy_mass(CANTERA.gas{WISEPHX}));
369          z= dH - Hadditional;
370       end %if CANTERA.DP==1
```

```
371 %       dX1 = XAirin + Xg - XAirFC - XFuelFC,
372 %       Xg,
373
374 %       if abs(XAirin + Xg - XAirFC - XFuelFC)>1E-10
375 %          dX1 = XAirin + Xg - XAirFC - XFuelFC;
376 %       end
377 %       disp('Durring Solver'), dX2 = XAirin + Xg - XAirFC - XFuelFC;
378 %       disp(['XAirFC=',num2str(XAirFC),' XFuelFC=',num2str(XFuelFC),' Xg=',↙
num2str(Xg),' dX=',num2str(dX2)]);
379 %       Xg,
380 %       z,
381
382    end %solve_FC_Fuel
383 %%
384    function z=solve_FC_HX_Eff(HX_Effg)
385       u = [XPComb; TPComb; PPComb; WISEPComb;  XAirin; TAirin; PAirin;↙
WISEAirin];
386       %[sys,x0,str,ts] = sfun_HeatExchanger(t,x,u,flag,n_HX,   ploss_hot,ploss_cold,↙
WISE_hot,WISE_cold,TCo);
387       [sys,x0,str,ts] = sfun_HeatExchanger(t,x,u, 30 ,HX_Effg,Ploss_HX, 0       ,↙
WISEPHX ,WISEAirHX,1000);
388       TPHX = sys(2); TAirHX = x0(2);
389       %Is dH across equal Hadd
390       dH = XAirin * enthalpy_mass(CANTERA.gas{WISEAirHX}) - HAirin;
391       z = Hadditional - dH;
392    end % solve_FC_HX_Eff
393
394 %%
395 %    XAcross,
396 %    disp(['XAirin = ',num2str(XAirin),' XFuelin = ',num2str(XFuel),' XAcross = ',↙
num2str(XAcross)]),
397 %    disp(['TAirFC = ',num2str(TAirFC)]);
398
399    if (flag==9 || flag==99)
400       if CANTERA.DATA==1
401          % detailed info on Fuel cell, Rec, fuel utilization, HX
402          % effectiveness, (dVJ + dVT) * JFC + Vo + dVP + dVoxygen + dVfuel)
403          % Ploss_anode, Ploss_cathode, Ploss_combustor, Ploss_HX
404          Z = [rec;n_Fuel;HX_Eff;Ploss_anode;Ploss_cathode;Ploss_combustor;↙
Ploss_HX;...
405             JFC;VFC;Vo;dVJ_J;dVT_J;dVP;dVoxygen;dVfuel];
406          assignin('base',['FC_Params_',num2str(WISEPHX)],Z);
407
408          H = XAirHX.* enthalpy_mass(CANTERA.gas{WISEAirHX});
409          S = XAirHX.* enthalpy_mass(CANTERA.gas{WISEAirHX});
410          dH= H - XAirin * enthalpy_mass(CANTERA.gas{WISEAirin});
411          Z = [XAirin;TAirin;PAirin;WISEAirin; XAirHX;TAirHX;PAirHX;WISEAirHX; H;S;↙
```

```
HX_Eff;dH;0;0;0];
412          assignin('base',['FCAirHX_',num2str(WISEAirHX)],Z);
413
414          % XTPW_in, XTPW_out, HS, Voltage, JFC, WorkFC 0 0
415          H = XAirFC.* enthalpy_mass(CANTERA.gas{WISEAirFC});
416          S = XAirFC.* enthalpy_mass(CANTERA.gas{WISEAirFC});
417          Z = [XAirin;TAirHX;PAirHX;WISEAirHX; XAirFC;TAirFC;PAirFC;WISEAirFC; H;S;↙
Voltage; JFC; WorkFC;0;0];
418          assignin('base',['FCAirFC_',num2str(WISEAirFC)],Z);
419
420          H = XPComb.* enthalpy_mass(CANTERA.gas{WISEPComb});
421          S = XPComb.* enthalpy_mass(CANTERA.gas{WISEPComb});
422          Z = [XAirFC;XFuelFC;WISEAirFC;WISEFuelFC; XPComb;TPComb;PPComb;↙
WISEPComb; H;S; 0;0;0;0;0 ];
423          assignin('base',['FCPComb_',num2str(WISEPComb)],Z);
424
425          H = XPHX.* enthalpy_mass(CANTERA.gas{WISEPHX});
426          S = XPHX.* enthalpy_mass(CANTERA.gas{WISEPHX});
427          dH= H - XPComb * enthalpy_mass(CANTERA.gas{WISEPComb});
428          Z = [XPComb;TPComb;PPComb;WISEPComb; XPHX;TPHX;PPHX;WISEPHX; H;S;↙
HX_Eff;dH;0;0;0];
429          assignin('base',['FCPHX_',num2str(WISEPHX)],Z);
430
431          H = XFuelReform.* enthalpy_mass(CANTERA.gas{WISEFuelReform});
432          S = XFuelReform.* enthalpy_mass(CANTERA.gas{WISEFuelReform});
433          Z = [0;0;0;0; XFuelReform;TFuelReform;PFuelReform;WISEFuelReform; H;S;0;↙
0;0;0;0];
434          assignin('base',['FCFuelReform_',num2str(WISEFuelReform)],Z);
435
436          H = XFuelFC.* enthalpy_mass(CANTERA.gas{WISEFuelFC});
437          S = XFuelFC.* enthalpy_mass(CANTERA.gas{WISEFuelFC});
438          Z = [XFuelReform;TFuelReform;PFuelReform;WISEFuelReform; XFuelFC;↙
TFuelFC;PFuelFC;WISEFuelFC; H;S; Voltage;JFC;WorkFC;0;0];
439          assignin('base',['FCFuelFC_',num2str(WISEFuelFC)],Z);
440
441          H = Xrec1.* enthalpy_mass(CANTERA.gas{WISErec1});
442          S = Xrec1.* enthalpy_mass(CANTERA.gas{WISErec1});
443          Z = [0;0;0;0; Xrec1;Trec1;Prec1;WISErec1; H;S; 0;0;0;0;0];
444          assignin('base',['FCrec1_',num2str(WISErec1)],Z);
445
446          H = Xrec2.* enthalpy_mass(CANTERA.gas{WISErec2});
447          S = Xrec2.* enthalpy_mass(CANTERA.gas{WISErec2});
448          Z = [0;0;0;0; Xrec2;Trec2;Prec2;WISErec2; H;S; 0;0;0;0;0];
449          assignin('base',['FCrec2_',num2str(WISErec2)],Z);
450
451
452          H = XFuel.* enthalpy_mass(CANTERA.gas{WISEFuelin});
```

```
453          S = XFuel.* enthalpy_mass(CANTERA.gas{WISEFuelin});
454          Z = [0;0;0;0; XFuel;TFuelin;PFuelin;WISEFuelin; H;S; 0;0;0;0;0];
455          assignin('base',['FCFuelin_',num2str(WISEFuelin)],Z);
456
457          H = XAirin.* enthalpy_mass(CANTERA.gas{WISEAirin});
458          S = XAirin.* enthalpy_mass(CANTERA.gas{WISEAirin});
459          Z = [0;0;0;0; XAirin;TAirin;PAirin;WISEAirin; H;S; 0;0;0;0;0];
460          assignin('base',['FCAirin_',num2str(WISEAirin)],Z);
461
462          % Species
463          Z = [XAirHX; TAirHX; PAirHX; WISEAirHX; XAirHX.*massFractions(CANTERA.↙
gas{WISEAirHX})];
464          assignin('base',['FCP_AirHX_',num2str(WISEAirHX)],Z);
465
466          Z = [XFuel; TFuelin; PFuelin; WISEFuelin; XFuel.*massFractions(CANTERA.gas↙
{WISEFuelin})];
467          assignin('base',['FCP_Fuelin_',num2str(WISEFuelin)],Z);
468
469          Z = [XFuelReform; TFuelReform; PFuelReform; WISEFuelReform;↙
XFuelReform.*massFractions(CANTERA.gas{WISEFuelReform})];
470          assignin('base',['FCP_FuelReform_',num2str(WISEFuelReform)],Z);
471
472          Z = [XFuelFC; TFuelFC; PFuelFC; WISEFuelFC; XFuelFC.*massFractions↙
(CANTERA.gas{WISEFuelFC})];
473          assignin('base',['FCP_FuelFC_',num2str(WISEFuelFC)],Z);
474
475          Z = [XAirFC; TAirFC; PAirFC; WISEAirFC; XAirFC.*massFractions(CANTERA.↙
gas{WISEAirFC})];
476          assignin('base',['FCP_AirFC_',num2str(WISEAirFC)],Z);
477
478          Z = [XPComb; TPComb; PPComb; WISEPComb; XPComb.*massFractions↙
(CANTERA.gas{WISEPComb})];
479          assignin('base',['FCP_PComb_',num2str(WISEPComb)],Z);
480
481          Z = [XPHX; TPHX; PPHX; WISEPHX; XPHX.*massFractions(CANTERA.gas↙
{WISEPHX})];
482          assignin('base',['FCP_PHX_',num2str(WISEPHX)],Z);
483
484       end
485    end
486 %    O2frac = avgmolFracO2
487 %    sys=[dX,TAirHX,TFC,Trec,TPComb,TPHX,Work];
488    sys = [XAirHX,TAirHX, PAirHX, WISEAirHX,...
489       XAirFC, TAirFC, PAirFC, WISEAirFC,...
490       XPComb, TPComb, PPComb, WISEPComb,...
491       XPHX, TPHX, PPHX, WISEPHX,...
492       XFuelReform, TFuelReform, PFuelReform, WISEFuelReform,...
```

```
493       XFuelFC, TFuelFC, PFuelFC, WISEFuelFC,...
494       Xrec1, Trec1, Prec1, WISErec1,...
495       Xrec2, Trec2, Prec2, WISErec2,...
496       XFuel, WorkFC, Voltage, JFC];
497
498 %    Ebalance = HAirin + XFuel*hFuelin + WorkFC*1000 - XPHX * enthalpy_mass↙
(CANTERA.gas{WISEPHX}),
499
500 %% Fuel Cell Fuel Calcs Copied from if FC_Type
501    function FC_Fuel_Calcs()
502    if FC_Type==1  %SOFC
503       MolsAcrossAvailable(CANTERA.I.O2) = MolsAirin(CANTERA.I.O2);
504       molsAcrossNeeded(CANTERA.I.O2) = molsFuelin' * CANTERA.O2_NEEDED;
505       MaxFuel = min(MolsAcrossAvailable ./ molsAcrossNeeded);
506    % Assume there will always be enough O2 and Calculate FuelFC Flows
507       molsAcross = molsAcrossNeeded .* n_Fuel;
508       mfAcross = molsAcross .* CANTERA.KG_KMOL;
509       xAcross = sum(mfAcross);
510       molsFuelFC = molsFuelin + molsAcross;
511       mfFuelFC = molsFuelFC .* CANTERA.KG_KMOL;
512       xFuelFC = xFuelin + xAcross;
513 %% SOFC Combustion at the exit of Fuel Cell Stack
514       set(CANTERA.gas{WISEFuelFC},'T',TFuelFC,'P',PFuelFC*oneatm,'X',molsFuelFC);
515       equilibrate(CANTERA.gas{WISEFuelFC},'TP'); % Gibbs Minimum
516 %       equilibrate_basic(WISEFuelFC,'TP');  % Basic Combustion
517    % Double Check conservation of Elements
518 %       mfFuelFC2= massFractions(CANTERA.gas{WISEFuelFC}),
519 %       atomFC2= CANTERA.ELEM * moleFractions(CANTERA.gas{WISEFuelFC}),
520       mfFuelFC = xFuelFC .* massFractions(CANTERA.gas{WISEFuelFC});
521       molsFuelFC = mfFuelFC ./ CANTERA.KG_KMOL;
522       hFuelFC = enthalpy_mass(CANTERA.gas{WISEFuelFC}); % Without an X↙
(assuming 1), it is used in solving loop
523
524    % Set recycle states
525       Trec1 = CellTemp;
526       Trec2 = CellTemp;
527       Prec1 = PFuelFC;
528       Prec2 = PFuelin *(1+0.05); % 5% more pressure
529       set(CANTERA.gas{WISErec1},'T',Trec1,'P',Prec1*oneatm,'X',molsFuelFC);
530       set(CANTERA.gas{WISErec2},'T',Trec2,'P',Prec2*oneatm,'X',molsFuelFC);
531
532       xrec1 = xFuelFC * rec/(1-rec); % Flow based on FC boundry layer
533       mfrec1 = mfFuelFC .* (rec / (1-rec));
534       hrec1 = xrec1 * enthalpy_mass(CANTERA.gas{WISErec1});
535    % Rec 2 is unused/not needed in SOFC
536
537    % Set Fuel Reform State
```

```
538       mfFuelReform = mfFuelin + mfrec1;
539       molsFuelReform = mfFuelReform ./ CANTERA.KG_KMOL;
540       hFuelReform = hFuelin + hrec1;
541       xFuelReform = xFuelin + xrec1;
542 %       mfFuelReform = mfFuelFC ./ (1-rec) - mfAcross; % Double Check
543 %       xFuelReform2 = xFuelFC / (1-rec) - xAcross;
544 %% SOFC Combustion at Fuel Reform begining of stack enterance
545       set(CANTERA.gas{WISEFuelReform},'H',hFuelReform / xFuelReform,'P',↙
PFuelReform*oneatm,'X',molsFuelReform);
546 %       equilibrate(CANTERA.gas{WISEFuelReform},'HP'); % Gibbs Minimume
547       equilibrate_basic(WISEFuelReform,'HP'); % Basic Combustion
548       TFuelReform = temperature(CANTERA.gas{WISEFuelReform});
549     % Unneeded in rest of loop
550 %       mfFuelReform = xFuelReform .* massFractions(CANTERA.gas↙
{WISEFuelReform});
551 %       molsFuelReform = mfFuelReform ./ CANTERA.KG_KMOL;
552 %% SOFC VI curve: V(mV)= (dVJ + dVT) *J(mA/cm^2) + Vo + dVP + dVoxygen +↙
dVfuel
553       dVJ = -0.6196; %From FC Handbook 7-22: P=5atm  mV*cm^2/mA
554       Vo  = 910.76; %From FC Handbook 7-22: P=5atm  mV
555       dVP = 59 * log10(PAirFC/5); %From FC Handbook 7-22  mV
556   %     dVT = 0; % Since cell is held at 1000C  mV*cm^2/mA
557       % From FC Handbook 7-11 through 7-13
558       if TAirFC >= 900+273.15, dVT = 0.008 * (TAirFC-1273.15);
559       elseif TAirFC >= 800+273.15, dVT = 0.04 * (TAirFC-1273.15);
560       elseif TAirFC >= 650+273.15, dVT = 1.3 * (TAirFC-1273.15);
561       else dVT = 1.3 * (TAirFC-1273.15); disp('TAirFC is below 650C!!');
562       end %if
563       molFracO2in = moleFraction(CANTERA.gas{WISEAirin},'O2');
564       % dVoxygen set in solver loop based upon O2 utilization
565       molFracFuelReform = moleFractions(CANTERA.gas{WISEFuelReform});
566       molFracFuelFC = moleFractions(CANTERA.gas{WISEFuelFC});
567       avgmolFracH2  = (molFracFuelFC(CANTERA.I.H2) + molFracFuelReform↙
(CANTERA.I.H2))/2;
568       avgmolFracH2O = (molFracFuelFC(CANTERA.I.H2O) + molFracFuelReform↙
(CANTERA.I.H2O))/2;
569       dVfuel = 172 * log10((avgmolFracH2/avgmolFracH2O) / (1.06212833)); %mV↙
FC Handbook 7-27
570   %   dVfuel2 = 172 * log10((avgmolFracH2/avgmolFracH2O) / (0.957721467)); %↙
mV FC Handbook 7-27
571   end %FC_Type
572
573   end %FC_Fuel_Calcs
574
575 end %mdlOutputs
576
577 %% mdlTerminate
```

```
578 function [sys]=mdlTerminate(t,x,u,flag, FC_Type, n_Fuel, rec, Ploss_anode,↙
Ploss_cathode, Ploss_combustor, Ploss_HX,  FC_Volt, HX_Eff, CellArea, CellTemp,↙
FCExitTemp, WISEAirHX, WISEAirFC, WISEPComb, WISEPHX, WISEFuelReform, WISEFuelFC,↙
WISErec1, WISErec2);
579     sys = [];
580 end %mdlTerminate
581 %% Equilibrate Basic
582 function equilibrate_basic(WISE,Condition)
583     global CANTERA
584     molMix = moleFractions(CANTERA.gas{WISE});
585     atomMix = CANTERA.ELEM * molMix;  % O H C N Ar
586     molProd = zeros(size(molMix));
587 %% C + O -> CO
588     X = min(atomMix(1),atomMix(3));
589     atomMix(1) = atomMix(1) - X;
590     atomMix(3) = atomMix(3) - X;
591     molProd(CANTERA.I.CO) = molProd(CANTERA.I.CO) + X;
592 %% CO + 2H + 2O -> CO2 + H2O
593     X = min([atomMix(1)/2, atomMix(2)/2, molProd(CANTERA.I.CO)]);
594     atomMix(1) = atomMix(1) - 2*X;
595     atomMix(2) = atomMix(2) - 2*X;
596     molProd(CANTERA.I.CO) = molProd(CANTERA.I.CO) - X;
597     molProd(CANTERA.I.CO2) = molProd(CANTERA.I.CO2) + X;
598     molProd(CANTERA.I.H2O) = molProd(CANTERA.I.H2O) + X;
599 %% CO + O -> CO2
600     X = min(atomMix(1), molProd(CANTERA.I.CO));
601     atomMix(1) = atomMix(1) - X;
602     molProd(CANTERA.I.CO) = molProd(CANTERA.I.CO) - X;
603     molProd(CANTERA.I.CO2) = molProd(CANTERA.I.CO2) + X;
604 %% 2H + O -> H2O
605     X = min(atomMix(1), atomMix(2)/2);
606     atomMix(1) = atomMix(1) - X;
607     atomMix(2) = atomMix(2) - 2*X;
608     molProd(CANTERA.I.H2O) = molProd(CANTERA.I.H2O) + X;
609 %% C + 4H -> CH4
610     X = min(atomMix(3), atomMix(2)/4);
611     atomMix(3) = atomMix(3) - X;
612     atomMix(2) = atomMix(2) - 4*X;
613     molProd(CANTERA.I.CH4) = molProd(CANTERA.I.CH4) + X;
614
615 %  Unneeded additional pieces, because they have not yet been assigned.
616 %% 2H -> H2
617     molProd(CANTERA.I.H2) = atomMix(2)/2; % + molProd(CANTERA.I.H2)
618 %     atomMix(2) = 0;
619 %% 2O -> O2
620     molProd(CANTERA.I.O2) = atomMix(1)/2 ; % + molProd(CANTERA.I.O2)
621 %     atomMix(1) = 0;
```

```
622 %% 2N -> N2
623     molProd(CANTERA.I.N2) = atomMix(4)/2 ; % + molProd(CANTERA.I.N2)
624 %     atomMix(4) = 0;
625 %% Ar
626     molProd(CANTERA.I.AR) = atomMix(5) ; % + molProd(CANTERA.I.AR)
627 %     atomMix(5) = 0;
628 %     atomMix2 = CANTERA.ELEM * molProd,  % O H C N Ar
629     P = pressure(CANTERA.gas{WISE});
630 %     if strcmp(Condition,'TP'), T = temperature(CANTERA.gas{WISE});
631 %     elseif strcmp(Condition,'HP'), h = enthalpy_mass(CANTERA.gas{WISE});
632 %     end
633     if strcmp(Condition,'TP'), set(CANTERA.gas{WISE},'T',temperature(CANTERA.gas{WISE}),'P',P,'X', molProd);
634     elseif strcmp(Condition,'HP'), set(CANTERA.gas{WISE},'H',enthalpy_mass(CANTERA.gas{WISE}),'P',P,'X', molProd);
635     end
636 end %equilibrium_basic
```

D.8. Fuel Cell Basic (One Exit Stream)

/Users/winstonburbank/Documents/Matlab Work/P.../sfun_Fuel_Cell_Basic1.m 1 of 3

```
 1 %sfun_Fuel_Cell_Basic_1out will compute the steady state out put from a
 2 %fuel cell assuming only one combusted stream exits the fuel cell.
 3 % Given: Fuel flow, fuel utilization and fuel cell voltage
 4
 5 function [sys,x0,str,ts] = sfun_Fuel_Cell_Basic1(t,x,u,flag, FC_Type, n_Fuel, Ploss,↙
FC_Volt, WISE)
 6 global CANTERA
 7 switch flag,
 8  case 0,  [sys,x0,str,ts]=mdlInitializeSizes(8,2);
 9          CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
10          set(CANTERA.gas{WISE},'T',1100,'P',3*oneatm,'X','CO2:0.033, H2O:0.067, N2:↙
0.73, O2:0.17');
11  case 1,   sys=mdlDerivatives(t,x,u);
12  case 2,   sys=mdlUpdate(t,x,u);
13  case 3,   sys=mdlOutputs(t,x,u,FC_Type, n_Fuel, Ploss, FC_Volt, WISE);
14  case 10,  sys=mdlOutputs(t,x,u,FC_Type, n_Fuel, Ploss, FC_Volt, WISE);   x0=0;↙
str=0; ts=0;
15  case 4,   sys=mdlGetTimeOfNextVarHit(t,x,u);
16  case 9,   sys=mdlTerminate(t,x,u);
17   otherwise,   error(['Unhandled flag = ',num2str(flag)]);
18 end %switch
19 end %sfun
20
21 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
22 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
23 sizes = simsizes;
24 sizes.NumContStates  = 0;
25 sizes.NumDiscStates  = 0;
26 sizes.NumOutputs     = out;
27 sizes.NumInputs      = in;
28 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs=1
29 sizes.NumSampleTimes = 1;  % at least one sample time is needed
30 sys = simsizes(sizes);
31 x0  = [];  % initialize the initial conditions
32 str = [];  % str is always an empty matrix
33 ts  = [0 0];  % initialize the array of sample times
34 end %mdlInitializeSizes
35
36 function sys=mdlDerivatives(t,x,u)
37 sys = [];
38 end %mdlDerivatives
39
40 function sys=mdlUpdate(t,x,u)
41 sys = [];
42 end %mdlUpdate
43
44
```

```
45 function [sys]=mdlOutputs(t,x,u, FC_Type, n_Fuel, Ploss, FC_Volt, WISE)
46 global CANTERA
47 %%Inputs%%
48 XFuelin = u(1); TFuelin = u(2); PFuelin = u(3); WISEFuelin = u(4); %Fuel
49 XAirin  = u(5); TAirin  = u(6); PAirin  = u(7); WISEAirin  = u(8); %Air
50
51 P3 = min([PFuelin,PAirin]) * (1-Ploss);
52 X3 = XFuelin + XAirin;
53 Work = 0;
54 if X3==0 || XFuelin==0,
55    set(CANTERA.gas{WISE},'T',TAirin,'P',P3*oneatm,'Y',massFractions(CANTERA.gas{WISEAirin}));
56 elseif XAirin==0,
57    set(CANTERA.gas{WISE},'T',TFuelin,'P',P3*oneatm,'Y',massFractions(CANTERA.gas{WISEFuelin}));
58 else
59
60    % Need to add n_Fuel
61    MfFuelin= XFuelin * massFractions(CANTERA.gas{WISEFuelin}); %kg/s
62    HFuelin = XFuelin * enthalpy_mass(CANTERA.gas{WISEFuelin}); %W
63    MolsFuelin = MfFuelin ./ CANTERA.KG_KMOL; %kMols/s
64
65    MfAirin = XAirin * massFractions(CANTERA.gas{WISEAirin}); %kg/s
66    HAirin = XAirin * enthalpy_mass(CANTERA.gas{WISEAirin}); %W
67    MolsAirin = MfAirin ./ CANTERA.KG_KMOL; %kMols/s
68
69    Across = zeros(CANTERA.NSP,1);
70    if FC_Type==1 %SOFC
71 %       AcrossAvailable = MolsAirin(CANTERA.I.O2);
72 %       AcrossNeeded = MolsFuelin' * CANTERA.O2_NEEDED * n_Fuel;
73    Across(CANTERA.I.O2) = min(MolsAirin(CANTERA.I.O2) , MolsFuelin' * CANTERA.O2_NEEDED * n_Fuel);%kMols/s
74    end
75
76    I = Across' * CANTERA.ELECTRONS * 96485 * 1000; %Amps
77    Work = -FC_Volt * I; %Watts
78    H3 = HFuelin + HAirin + Work;
79
80    set(CANTERA.gas{WISE},'H',H3/X3,'P',P3*oneatm,'Y',(MfFuelin + MfAirin)./X3);
81    equilibrate(CANTERA.gas{WISE},'HP');
82 end %if
83
84 %  Need to add hloss
85 T3=temperature(CANTERA.gas{WISE});
86 Work = Work / 1000;  %kW
87 sys=[T3,Work];
88 end %mdlOutputs
```

```
89
90
91 function sys=mdlGetTimeOfNextVarHit(t,x,u)
92 sampleTime = 1;    % Example, set the next hit to be one second later.
93 sys = t + sampleTime;
94 end %mdlGetTimeOfNextVarHit
95
96 function sys=mdlTerminate(t,x,u)
97 sys = [];
98 end %mdlTerm
```

D.9. Fuel Cell Basic 2 (Two Exit Streams)

/Users/winstonburbank/Documents/Matlab Work/P.../sfun_Fuel_Cell_Basic2.m 1 of 4

```
 1 %sfun_Fuel_Cell_Basic_1out will compute the steady state out put from a
 2 %fuel cell assuming only one combusted stream exits the fuel cell.
 3 % Given: Fuel flow, fuel utilization and fuel cell voltage
 4
 5 function [sys,x0,str,ts] = sfun_Fuel_Cell_Basic2(t,x,u,flag, FC_Type, n_Fuel, Ploss,↙
FC_Volt, WISEFUEL, WISEAIR)
 6 global CANTERA
 7 switch flag,
 8  case 0,   [sys,x0,str,ts]=mdlInitializeSizes(8,3);
 9          CANTERA.gas{WISEFUEL} = importPhase('Basic.cti',CANTERA.GAS);
10          set(CANTERA.gas{WISEFUEL},'T',1100,'P',3*oneatm,'X','CO2:0.50, H2O:0.40,↙
H2:0.10');
11          CANTERA.gas{WISEAIR} = importPhase('Basic.cti',CANTERA.GAS);
12          set(CANTERA.gas{WISEAIR},'T',1100,'P',3*oneatm,'X','AR:0.10 N2:0.80, O2:↙
0.10');
13  case 1,   sys=mdlDerivatives(t,x,u);
14  case 2,   sys=mdlUpdate(t,x,u);
15  case 3,   sys=mdlOutputs(t,x,u,FC_Type, n_Fuel, Ploss, FC_Volt,  WISEFUEL,↙
WISEAIR);
16  case 10,  sys=mdlOutputs(t,x,u,FC_Type, n_Fuel, Ploss, FC_Volt,  WISEFUEL,↙
WISEAIR);   x0=0; str=0; ts=0;
17  case 4,   sys=mdlGetTimeOfNextVarHit(t,x,u);
18  case 9,   sys=mdlTerminate(t,x,u);
19   otherwise,   error(['Unhandled flag = ',num2str(flag)]);
20 end %switch
21 end %sfun
22
23 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
24 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
25 sizes = simsizes;
26 sizes.NumContStates  = 0;
27 sizes.NumDiscStates  = 0;
28 sizes.NumOutputs     = out;
29 sizes.NumInputs      = in;
30 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs=1
31 sizes.NumSampleTimes = 1;  % at least one sample time is needed
32 sys = simsizes(sizes);
33 x0  = [];  % initialize the initial conditions
34 str = [];  % str is always an empty matrix
35 ts  = [0 0];  % initialize the array of sample times
36 end %mdlInitializeSizes
37
38 function sys=mdlDerivatives(t,x,u)
39 sys = [];
40 end %mdlDerivatives
41
42 function sys=mdlUpdate(t,x,u)
```

```
 43 sys = [];
 44 end %mdlUpdate
 45
 46
 47 function [sys]=mdlOutputs(t,x,u, FC_Type, n_Fuel, Ploss, FC_Volt, WISEFUEL,↙
WISEAIR)
 48 global CANTERA
 49 %%inputs%%
 50 XFuelin = u(1); TFuelin = u(2); PFuelin = u(3); WISEFuelin = u(4); %Fuel
 51 XAirin  = u(5); TAirin  = u(6); PAirin  = u(7); WISEAirin  = u(8); %Air
 52
 53 P3 = min([PFuelin,PAirin]) * (1-Ploss);
 54 TFC = temperature(CANTERA.gas{WISEAIR});
 55 % X3 = XFuelin + XAirin;
 56
 57 if XFuelin==0 || XAirin==0,
 58    set(CANTERA.gas{WISEFUEL},'T',temperature(CANTERA.gas{WISEFuelin}),'P',↙
P3*oneatm,'Y',massFractions(CANTERA.gas{WISEFuelin}));
 59    set(CANTERA.gas{WISEAIR},'T',temperature(CANTERA.gas{WISEAirin}),'P',↙
P3*oneatm,'Y',massFractions(CANTERA.gas{WISEAirin}));
 60    Work = 0; dX = 0;
 61 else
 62    % Need to add n_Fuel
 63    MfFuelin= XFuelin * massFractions(CANTERA.gas{WISEFuelin}); %kg/s
 64    HFuelin = XFuelin * enthalpy_mass(CANTERA.gas{WISEFuelin}); %W
 65    MolsFuelin = MfFuelin ./ CANTERA.KG_KMOL; %kMols/s
 66
 67    MfAirin = XAirin * massFractions(CANTERA.gas{WISEAirin}); %kg/s
 68    HAirin = XAirin * enthalpy_mass(CANTERA.gas{WISEAirin}); %W
 69    MolsAirin = MfAirin ./ CANTERA.KG_KMOL; %kMols/s
 70
 71    Hin = HFuelin + HAirin;
 72
 73    Across = zeros(CANTERA.NSP,1);
 74    if FC_Type==1 %SOFC
 75    Across(CANTERA.I.O2) = min(MolsAirin(CANTERA.I.O2), MolsFuelin' * CANTERA.↙
O2_NEEDED * n_Fuel); %kMols/s
 76    end
 77    MfAcross = Across .* CANTERA.KG_KMOL; %possitive is air to fuel side.
 78
 79    I = Across' * CANTERA.ELECTRONS * 96485 * 1000; %Amps
 80    Work = -FC_Volt * I; %Watts
 81
 82    % New Mass Flows
 83    dX = sum(MfAcross);
 84    MfAirFC = MfAirin - MfAcross;   XAirFC = XAirin - dX; %sum(MfAirFC);
 85    MfFuelFC = MfFuelin + MfAcross; XFuelFC = XFuelin + dX; %sum(Mffuel);
```

```
 86
 87    set(CANTERA.gas{WISEAIR},'T',TFC,'P',P3*oneatm,'Y',MfAirFC);
 88    set(CANTERA.gas{WISEFUEL},'T',TFC,'P',P3*oneatm,'Y',MfFuelFC);
 89
 90    options = optimset('Display','off'); % Turn off Display
 91    TFC = fsolve(@FC,TFC,options);
 92
 93 end %if
 94
 95    function Z=FC(TFCg)
 96      set(CANTERA.gas{WISEAIR},'T',TFCg,'P',P3*oneatm);
 97      set(CANTERA.gas{WISEFUEL},'T',TFCg,'P',P3*oneatm);
 98       if FC_Type==1
 99 %          equilibrate(CANTERA.gas{WISEFUEL},'TP');
100          equilibrate_basic(WISEFUEL,'TP');
101       end
102       HAirFC  = XAirFC  * enthalpy_mass(CANTERA.gas{WISEAIR});
103       HFuelFC = XFuelFC * enthalpy_mass(CANTERA.gas{WISEFUEL});
104       H3 = HAirFC + HFuelFC - Work;
105       Z= H3 - Hin;
106    end
107
108 Work = Work / 1000;  %kW
109 sys=[dX,TFC,Work];
110 end %mdlOutputs
111
112 function equilibrate_basic(WISE,Condition)
113    global CANTERA
114    P = pressure(CANTERA.gas{WISE});
115    if strcmp(Condition,'TP'), T = temperature(CANTERA.gas{WISE});
116    elseif strcmp(Condition,'HP'), h = enthalpy_mass(CANTERA.gas{WISE});
117    end
118    molMix = moleFractions(CANTERA.gas{WISE});
119    atomMix = CANTERA.ELEM * molMix;  % O H C N Ar
120    molProd = zeros(size(molMix));
121 %% C + O -> CO
122    X = min(atomMix(1),atomMix(3));
123    atomMix(1) = atomMix(1) - X;
124    atomMix(3) = atomMix(3) - X;
125    molProd(CANTERA.I.CO) = molProd(CANTERA.I.CO) + X;
126 %% CO + 2H + 2O -> CO2 + H2O
127    X = min([atomMix(1)/2, atomMix(2)/2, molProd(CANTERA.I.CO)]);
128    atomMix(1) = atomMix(1) - 2*X;
129    atomMix(2) = atomMix(2) - 2*X;
130    molProd(CANTERA.I.CO) = molProd(CANTERA.I.CO) - X;
131    molProd(CANTERA.I.CO2) = molProd(CANTERA.I.CO2) + X;
132    molProd(CANTERA.I.H2O) = molProd(CANTERA.I.H2O) + X;
```

```
133 %% CO + O -> CO2
134     X = min(atomMix(1), molProd(CANTERA.I.CO));
135     atomMix(1) = atomMix(1) - X;
136     molProd(CANTERA.I.CO) = molProd(CANTERA.I.CO) - X;
137     molProd(CANTERA.I.CO2) = molProd(CANTERA.I.CO2) + X;
138 %% 2H + O -> H2O
139     X = min(atomMix(1), atomMix(2)/2);
140     atomMix(1) = atomMix(1) - X;
141     atomMix(2) = atomMix(2) - 2*X;
142     molProd(CANTERA.I.H2O) = molProd(CANTERA.I.H2O) + X;
143 %% C + 4H -> CH4
144     X = min(atomMix(3), atomMix(2)/4);
145     atomMix(3) = atomMix(3) - X;
146     atomMix(2) = atomMix(2) - 4*X;
147     molProd(CANTERA.I.CH4) = molProd(CANTERA.I.CH4) + X;
148 %% 2H -> H2
149     molProd(CANTERA.I.H2) = atomMix(2)/2; % + molProd(CANTERA.I.H2)
150 %% 2O -> O2
151     molProd(CANTERA.I.O2) = atomMix(1)/2 ; % + molProd(CANTERA.I.O2)
152 %       atomMix(1) = 0;
153 %% 2N -> N2
154     molProd(CANTERA.I.N2) = atomMix(4)/2 ; % + molProd(CANTERA.I.N2)
155 %       atomMix(4) = 0;
156 %% Ar
157     molProd(CANTERA.I.AR) = atomMix(5) ; % + molProd(CANTERA.I.AR)
158 %       atomMix(5) = 0;
159 %     atomMix2 = CANTERA.ELEM * molProd,  % C H C N Ar
160     if strcmp(Condition,'TP'), set(CANTERA.gas{WISE},'T',T,'P',P,'X', molProd);
161     elseif strcmp(Condition,'HP'), set(CANTERA.gas{WISE},'H',h,'P',P,'X', molProd);
162     end
163 end %equilibrium_basic
164
165
166 function sys=mdlGetTimeOfNextVarHit(t,x,u)
167 sampleTime = 1;    %  Example, set the next hit to be one second later.
168 sys = t + sampleTime;
169 end %mdlGetTimeOfNextVarHit
170
171 function sys=mdlTerminate(t,x,u)
172 sys = [];
173 end %mdlTerm
```

D.10. Gas Input

/Users/winstonburbank/Documents/Matlab Work/PHD Libr.../sfun_Gas_Input.m 1 of 2

```
 1 %This Sfun will Supply Gas flow at specified Global settings.
 2 function [sys,x0,str,ts] = sfun_Gas_Input(t,x,u,flag,Gas_Type,Gas_Flow,WISE,↙
Mf_T_P)%u= 1:Ambient Air  2:Supplied Fuel (3:Nitrogen
 3 global CANTERA
 4 switch flag;
 5   case 0,    [sys,x0,str,ts]=mdlInitializeSizes(0,4);
 6            if Gas_Type ~=3, CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.↙
GAS); end
 7   case 3,    sys=mdlOutputs(flag,Gas_Type,Gas_Flow,WISE,Mf_T_P);
 8   %case 4,    sys=t+.1;
 9   case 9,    %sys=mdlTerminate(t,x,u,flag,Gas_Type,Gas_Flow,WISE,Mf_T_P);
10            sys=mdlOutputs(flag,Gas_Type,Gas_Flow,WISE,Mf_T_P);
11   case {1,2,4},    sys=[];
12   case {10,30,99},   sys=mdlOutputs(flag,Gas_Type,Gas_Flow,WISE,Mf_T_P); x0=[];↙
str=[]; ts=[];
13    otherwise,    error(['Unhandled flag = ',num2str(flag)]);
14 end
15 %end %sfun_Gas_Input
16
17 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
18 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
19 sizes = simsizes;
20 sizes.NumContStates  = 0;
21 sizes.NumDiscStates  = 0;
22 sizes.NumOutputs     = out;
23 sizes.NumInputs      = in;
24 sizes.DirFeedthrough = 0;   % u is needed for the calc of outputs=1
25 sizes.NumSampleTimes = 1;   % at least one sample time is needed
26 sys = simsizes(sizes);
27 x0  = [];  % initialize the initial conditions
28 str = [];  % str is always an empty matrix
29 ts  = [0 0];  % initialize the array of sample times
30 %end %mdlInitializeSizes
31
32 function [X_T_P_W]=mdlOutputs(flag,Gas_Type,Gas_Flow,WISE,Mf_T_P)
33    global CANTERA
34
35    switch Gas_Type
36       case 0, %No Gasses
37          Gas_Flow=sum(Mf_T_P(1:end-2));
38          if Gas_Flow==0, Y=massFractions(CANTERA.AIR);
39          else          Y=Mf_T_P(1:end-2)'./Gas_Flow;
40          end
41          T=Mf_T_P(end-1);
42          P=Mf_T_P(end);
43       case 1, %Ambient Air
44          Y=massFractions(CANTERA.AIR);
```

```
45          T=temperature(CANTERA.AIR);
46          P=pressure(CANTERA.AIR);
47        case 2, %Supplied Fuel
48          Y=massFractions(CANTERA.FUEL);
49          T=temperature(CANTERA.FUEL);
50          P=pressure(CANTERA.FUEL);
51        case 3,
52          Y=massFractions(CANTERA.gas{WISE});
53          T=temperature(CANTERA.gas{WISE});
54          P=pressure(CANTERA.gas{WISE});
55        otherwise,
56
57    end
58
59    set(CANTERA.gas{WISE},'T',T,'P',P,'Y',Y);
60    if (flag==9 || flag==99) && CANTERA.DATA==1
61       X=Gas_Flow;
62       H = X*enthalpy_mass(CANTERA.gas{WISE});
63       S = X*entropy_mass(CANTERA.gas{WISE});
64       Z = [0;0;0;0;X;T;P/oneatm;WISE;H;S;0;0;0;0;0];
65       assignin('base',['Gas_Input_',num2str(WISE)],Z);
66       Z = [X;T;P/oneatm;WISE;X.*massFractions(CANTERA.gas{WISE})];
67       assignin('base',['Gas_Input_Species_',num2str(WISE)],Z);
68    end
69
70    X_T_P_W=[Gas_Flow;T;P/oneatm;WISE];
71
72 %end %mdlOutputs
73
74 function sys=mdlTerminate(t,x,u,flag,Gas_Type,Gas_Flow,WISE,Mf_T_P)
75 %    global CANTERA
76 %    Z = mdlOutputs(Gas_Type,Gas_Flow,WISE,Mf_T_P);
77 %    X=Z(1); T=Z(2); P=Z(3);
78 %    H = X*enthalpy_mass(CANTERA.gas{WISE});
79 %    S = X*entropy_mass(CANTERA.gas{WISE});
80 %    if CANTERA.DATA==1
81 %       Z = [0;0;0;0;X;T;P;WISE;H;S;0;0;0;0;0];
82 %       assignin('base',['Gas_Input_',num2str(WISE)],Z);
83 %       Z = [X;T;P;WISE;X.*massFractions(CANTERA.gas{WISE})];
84 %       assignin('base',['Gas_Input_Species_',num2str(WISE)],Z);
85 %    end
86    sys=[];
```

D.11. Gas Set State

/Users/winstonburbank/Documents/Matlab Work/PHD Library/sfun_Gas_Set.m 1 of 2

```
% Sets a gas to the input Temperature (K) Pressure(atm)

function [sys,x0,str,ts] = sfun_Gas_Set(t,x,u,flag,WISE)
global CANTERA
switch flag,
 case 0,  [sys,x0,str,ts]=mdlInitializeSizes(6,0);
          CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
          set(CANTERA.gas{WISE},'T',800,'P',1.5*oneatm,'X','CH4:1');
 case 1,  sys=mdlDerivatives(t,x,u);
 case 2,  sys=mdlUpdate(t,x,u);
 case 3,  sys=mdlOutputs(t,x,u,flag,WISE);
 case {10,30,99},  sys=mdlOutputs(t,x,u,flag,WISE);  x0=0; str=0; ts=0;
          sys = [u(1);u(5);u(6);WISE];
 case 4,  sys=mdlGetTimeOfNextVarHit(t,x,u);
 case 9,  %sys=mdlTerminate(t,x,u,WISE);
          sys=mdlOutputs(t,x,u,flag,WISE);
  otherwise,  error(['Unhandled flag = ',num2str(flag)]);
end %switch
%end %sfun

function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
% global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = out;
sizes.NumInputs      = in;
sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs=1
sizes.NumSampleTimes = 1;  % at least one sample time is needed
sys = simsizes(sizes);
x0  = [];  % initialize the initial conditions
str = [];  % str is always an empty matrix
ts  = [0 0];  % initialize the array of sample times
%end %mdlInitializeSizes

function sys=mdlDerivatives(t,x,u)
sys = [];
%end %mdlDerivatives

function sys=mdlUpdate(t,x,u)
sys = [];
%end %mdlUpdate


function [sys]=mdlOutputs(t,x,u,flag,WISE)
global CANTERA
%%Inputs%%
```

```
48 X1=u(1); T1=u(2); P1=u(3); WISEin=u(4);
49 T2=u(5); P2=u(6);
50 set(CANTERA.gas{WISE},'T',T2,'P',P2*oneatm,'Y',massFractions(CANTERA.gas{WISEin}));
51
52 if (flag==9 || flag==99) && CANTERA.DATA==1
53    Z = [X1;T2;P2;WISE];
54    assignin('base',['Gas_',num2str(WISE)],Z);
55 end
56
57 sys=[];
58 %end %mdlOutputs
59
60
61 function sys=mdlGetTimeOfNextVarHit(t,x,u)
62 sampleTime = 1;    % Example, set the next hit to be one second later.
63 sys = t + sampleTime;
64 %end %mdlGetTimeOfNextVarHit
65
66 function sys=mdlTerminate(t,x,u,WISE)
67 % global CANTERA
68 % X1=u(1); T1=u(2); P1=u(3); WISEin=u(4);
69 % T2=u(5); P2=u(6);
70 % if CANTERA.DATA==1
71 %    Z = [X1;T2;P2;WISE];
72 %    assignin('base',['Gas_',num2str(WISE)],Z);
73 % end%if
74 sys = [];
75 %end %mdlTerminate
76
```

D.12. Set Gas H (Enthalpy)

/Users/winstonburbank/Documents/Matlab Work/PHD Libr.../sfun_Gas_Set_H.m 1 of 2

```
1 % Sets a gas to the input Temperature (K) Pressure(atm)
2
3 function [sys,x0,str,ts] = sfun_Gas_Set_H(t,x,u,flag,WISE)
4 global CANTERA
5 switch flag,
6  case 0,   [sys,x0,str,ts]=mdlInitializeSizes(5,4);
7          CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
8          set(CANTERA.gas{WISE},'T',800,'P',1.5*oneatm,'X','CH4:1');
9  case 1,    sys=mdlDerivatives(t,x,u);
10  case 2,    sys=mdlUpdate(t,x,u);
11  case 3,    sys=mdlOutputs(t,x,u,flag,WISE);
12  case {10,30,99},   sys=mdlOutputs(t,x,u,flag,WISE);    x0=0; str=0; ts=0;
13          sys = [u(1);u(5);u(6);WISE];
14  case 4,    sys=mdlGetTimeOfNextVarHit(t,x,u);
15  case 9,    %sys=mdlTerminate(t,x,u,WISE);
16          sys=mdlOutputs(t,x,u,flag,WISE);
17   otherwise,    error(['Unhandled flag = ',num2str(flag)]);
18 end %switch
19 %end %sfun
20
21 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
22 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
23 sizes = simsizes;
24 sizes.NumContStates  = 0;
25 sizes.NumDiscStates  = 0;
26 sizes.NumOutputs     = out;
27 sizes.NumInputs      = in;
28 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs=1
29 sizes.NumSampleTimes = 1;  % at least one sample time is needed
30 sys = simsizes(sizes);
31 x0  = []; % initialize the initial conditions
32 str = []; % str is always an empty matrix
33 ts  = [0 0]; % initialize the array of sample times
34 %end %mdlInitializeSizes
35
36 function sys=mdlDerivatives(t,x,u)
37 sys = [];
38 %end %mdlDerivatives
39
40 function sys=mdlUpdate(t,x,u)
41 sys = [];
42 %end %mdlUpdate
43
44
45 function [sys]=mdlOutputs(t,x,u,flag,WISE)
46 global CANTERA
47 %%%Inputs%%%
```

```
48 X1=u(1); T1=u(2); P1=u(3); WISEin=u(4);
49 dH=u(5);
50 H2 = enthalpy_mass(CANTERA.gas{WISEin})-dH*1000/X1;
51 set(CANTERA.gas{WISE},'H',H2,'P',P1*oneatm,'Y',massFractions(CANTERA.gas↙
{WISEin}));
52 T2 = temperature(CANTERA.gas{WISE});
53 if (flag==9 || flag==99) && CANTERA.DATA==1
54    Z = [X1;T2;P1;WISE];
55    assignin('base',['Gas_',num2str(WISE)],Z);
56 end
57
58 sys=[X1,T2,P1,WISE];
59 %end %mdlOutputs
60
61
62 function sys=mdlGetTimeOfNextVarHit(t,x,u)
63 sampleTime = 1;    %  Example, set the next hit to be one second later.
64 sys = t + sampleTime;
65 %end %mdlGetTimeOfNextVarHit
66
67 function sys=mdlTerminate(t,x,u,WISE)
68 % global CANTERA
69 % X1=u(1); T1=u(2); P1=u(3); WISEin=u(4);
70 % T2=u(5); P2=u(6);
71 % if CANTERA.DATA==1
72 %    Z = [X1;T2;P2;WISE];
73 %    assignin('base',['Gas_',num2str(WISE)],Z);
74 % end%if
75 sys = [];
76 %end %mdlTerminate
77
```

D.13. Heat Exchanger

/Users/winstonburbank/Documents/Matlab Work/PH.../sfun_HeatExchanger.m 1 of 3

```
 1
 2 function [sys,x0,str,ts] = sfun_HeatExchanger(t,x,u,flag,n_HX,ploss_hot,ploss_cold,WISE_hot,WISE_cold,TCo)
 3 global CANTERA
 4 switch flag,
 5  case 0,    [sys,x0,str,ts]=mdlInitializeSizes(8,2);
 6          CANTERA.gas{WISE_hot} = importPhase('Basic.cti',CANTERA.GAS);
 7          CANTERA.gas{WISE_cold} = importPhase('Basic.cti',CANTERA.GAS);
 8          set(CANTERA.gas{WISE_hot},'T',1100,'P',1.3*oneatm,'X','CO2:0.1, H2O:0.18, N2:0.71, AR:0.01');
 9          set(CANTERA.gas{WISE_cold},'T',TCo,'P',8*oneatm,'X','O2:0.21, N2:0.78, AR:0.01');
10          CANTERA.update{WISE_cold}=[0;800;800];
11  case 1,    sys=mdlDerivatives(t,x,u);
12  case 2,    sys=[];%mdlUpdate(t,x,u,n_HX,ploss_hot,ploss_cold,WISE_hot,WISE_cold);
13  case 3,    sys=mdlOutputs(t,x,u,flag,n_HX,ploss_hot,ploss_cold,WISE_hot,WISE_cold);
14  case {10,30,99},   sys=mdlOutputs(t,x,u,flag,n_HX,ploss_hot,ploss_cold,WISE_hot,WISE_cold);
15          x0 =[u(5);sys(2);u(7)*(1-ploss_cold);WISE_cold];
16          sys=[u(1);sys(1);u(3)*(1-ploss_hot);WISE_hot];
17          str=[]; ts=[];
18  case 4,    sys=mdlGetTimeOfNextVarHit(t,x,u);
19  case 9,    %sys=mdlTerminate(t,x,u,n_HX,ploss_hot,ploss_cold,WISE_hot,WISE_cold);
20          sys=mdlOutputs(t,x,u,flag,n_HX,ploss_hot,ploss_cold,WISE_hot,WISE_cold);
21   otherwise,   error(['Unhandled flag = ',num2str(flag)]);
22 end %switch
23 %end %sfun
24
25 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
26 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
27 sizes = simsizes;
28 sizes.NumContStates  = 0;
29 sizes.NumDiscStates  = 0;
30 sizes.NumOutputs     = out;
31 sizes.NumInputs      = in;
32 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs
33 sizes.NumSampleTimes = 1;  % at least one sample time is needed
34 sys = simsizes(sizes);
35 x0  = []; % initialize the initial conditions
36 str = []; % str is always an empty matrix
37 ts  = [0 0]; % initialize the array of sample times
38 %end %mdlInitializeSizes
39
40 function sys=mdlDerivatives(t,x,u)
41 sys = [];
```

```
 42 %end %mdlDerivatives
 43
 44 function sys=mdlUpdate(t,x,u,n_HX,WISE_hot,WISE_cold)
 45 global CANTERA
 46 X=mdlOutputs(t,x,u,n_HX,WISE_hot,WISE_cold);
 47 CANTERA.update{WISE_cold}=[t;X(1);X(2)];
 48 sys = [];
 49 %end %mdlUpdate
 50
 51
 52 function [sys]=mdlOutputs(t,x,u,flag,n_HX,ploss_hot,ploss_cold,WISE_hot,WISE_cold)
 53 global CANTERA
 54 % if t == CANTERA.update{WISE_cold}(1)
 55 %%Inputs%%
 56 Xhot=u(1); Thot=u(2); Phot=u(3)*oneatm; WISE_hotin=u(4);
 57 Xcold=u(5); Tcold=u(6); Pcold=u(7)*oneatm; WISE_coldin=u(8);
 58
 59 % set(CANTERA.gas{WISE_hot},'T',Thot,'P',Phot,'Y',massFractions(CANTERA.gas{WISE_hotin}));
 60 % set(CANTERA.gas{WISE_cold},'T',Tcold,'P',Pcold,'Y',massFractions(CANTERA.gas{WISE_coldin}));
 61
 62 if Xhot>0 && Xcold>0
 63    %% Enthalpy in
 64    HhotThot = Xhot * enthalpy_mass(CANTERA.gas{WISE_hotin});
 65    set(CANTERA.gas{WISE_hot},'T',Tcold,'P',Phot,'Y',massFractions(CANTERA.gas{WISE_hotin}));
 66    HhotTcold = Xhot * enthalpy_mass(CANTERA.gas{WISE_hot});
 67
 68    HcoldTcold = Xcold * enthalpy_mass(CANTERA.gas{WISE_coldin});
 69    set(CANTERA.gas{WISE_cold},'T',Thot,'P',Pcold,'Y',massFractions(CANTERA.gas{WISE_coldin}));
 70    HcoldThot = Xcold * enthalpy_mass(CANTERA.gas{WISE_cold});
 71
 72    %% This Selects the lesser energy exchange and multiples by the
 73    % effectiveness
 74    d1 = HhotThot - HhotTcold;
 75    d2 = HcoldThot - HcoldTcold;
 76    if d1^2<d2^2, dH=d1*n_HX; else dH=d2*n_HX; end %if
 77    Hhot = HhotThot - dH;
 78    Hcold = HcoldTcold + dH;
 79
 80    set(CANTERA.gas{WISE_hot},'H',Hhot/Xhot,'P',Phot*(1-ploss_hot)); %Ploss???
 81    set(CANTERA.gas{WISE_cold},'H',Hcold/Xcold,'P',Pcold*(1-ploss_cold)); %Ploss???
 82 end %if Xhot && Xcold
 83
 84    Thot_out=temperature(CANTERA.gas{WISE_hot});
```

```
 85    Tcold_out=temperature(CANTERA.gas{WISE_cold});
 86
 87    if (flag==9 || flag==99) && CANTERA.DATA==1
 88       Hhot = Xhot * enthalpy_mass(CANTERA.gas{WISE_hot});
 89       Shot = Xhot * entropy_mass(CANTERA.gas{WISE_hot});
 90       dHhot = Hhot - Xhot * enthalpy_mass(CANTERA.gas{WISE_hotin});
 91       Z = [Xhot;Thot;Phot/oneatm;WISE_hotin; Xhot;Thot_out;Phot*(1-ploss_hot)↙
/oneatm;WISE_hot; Hhot;Shot;n_HX;dHhot;0;0;0];
 92       assignin('base',['HeatExchanger_',num2str(WISE_hot)],Z);
 93
 94       Hcold = Xcold * enthalpy_mass(CANTERA.gas{WISE_cold});
 95       Scold = Xcold * entropy_mass(CANTERA.gas{WISE_cold});
 96       dHcold = Hcold - Xcold * enthalpy_mass(CANTERA.gas{WISE_coldin});
 97       Z = [Xcold;Tcold;Pcold/oneatm;WISE_coldin; Xcold;Tcold_out;Pcold*(1-↙
ploss_cold)/oneatm;WISE_cold; Hcold;Scold;n_HX;dHcold;0;0;0];
 98       assignin('base',['HeatExchanger_',num2str(WISE_cold)],Z);
 99    end
100
101 sys=[Thot_out; Tcold_out];
102 %end %mdlOutputs
103
104
105 function sys=mdlGetTimeOfNextVarHit(t,x,u)
106 sampleTime = 1;    % Example, set the next hit to be one second later.
107 sys = t + sampleTime;
108 %end %mdlGetTimeOfNextVarHit
109
110 function sys=mdlTerminate(t,x,u,n_HX,ploss_hot,ploss_cold,WISE_hot,WISE_cold)
111 sys = [];
112 %end %mdlTerminate
113
```

D.14. InterCooler

/Users/winstonburbank/Documents/Matlab Work/PHD Lib.../sfun_Intercooler.m 1 of 3

```
 1
 2 function [sys,x0,str,ts] = sfun_Intercooler(t,x,u,flag,dT,ploss,WISE)
 3 global CANTERA
 4 switch flag,
 5  case 0,    [sys,x0,str,ts]=mdlInitializeSizes(8,1);
 6          CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
 7          set(CANTERA.gas{WISE},'T',1100,'P',1.3*oneatm,'X','O2:0.21, N2:0.78, AR:↙
0.01');
 8  case 1,    sys=mdlDerivatives(t,x,u);
 9  case 2,    sys=[];%mdlUpdate(t,x,u,n_HX,ploss_hot,ploss_cold,WISE_hot,WISE_cold);
10  case 3,    %disp(['Heat X: t=',num2str(t),'  u= ',num2str(u),'  WISE= ',num2str↙
(WISE_cold)]),
11           sys=mdlOutputs(t,x,u,flag,dT,ploss,WISE);
12  case {10,30,99},  sys=mdlOutputs(t,x,u,flag,dT,ploss,WISE);    x0=0; str=0; ts=0;
13          sys=[u(1);sys;u(3)*(1-ploss);WISE];
14  case 4,    sys=mdlGetTimeOfNextVarHit(t,x,u);
15  case 9,    %sys=mdlTerminate(t,x,u,flag,dT,ploss,WISE);
16          sys=mdlOutputs(t,x,u,flag,dT,ploss,WISE);
17   otherwise,    error(['Unhandled flag = ',num2str(flag)]);
18 end %switch
19 %end %sfun
20
21 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
22 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
23 sizes = simsizes;
24 sizes.NumContStates  = 0;
25 sizes.NumDiscStates  = 0;
26 sizes.NumOutputs     = out;
27 sizes.NumInputs      = in;
28 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs
29 sizes.NumSampleTimes = 1;  % at least one sample time is needed
30 sys = simsizes(sizes);
31 x0  = []; % initialize the initial conditions
32 str = []; % str is always an empty matrix
33 ts  = [0 0]; % initialize the array of sample times
34 %end %mdlInitializeSizes
35
36 function sys=mdlDerivatives(t,x,u)
37 sys = [];
38 %end %mdlDerivatives
39
40 function sys=mdlUpdate(t,x,u,flag,dT,ploss,WISE)
41 global CANTERA
42 X=mdlOutputs(t,x,u,n_HX,WISE_hot,WISE_cold);
43 CANTERA.update{WISE_cold}=[t;X(1);X(2)];
44 sys = [];
45 %end %mdlUpdate
```

```
46
47
48 function [sys]=mdlOutputs(t,x,u,flag,dT,ploss,WISE)
49 global CANTERA
50 % if t == CANTERA.update{WISE_cold}(1)
51 %%Inputs%%
52 Xhot=u(1); Thot=u(2); Phot=u(3)*oneatm; WISE_hotin=u(4); % Working fluid
53 Xcold=u(5); Tcold=u(6); Pcold=u(7)*oneatm; WISE_coldin=u(8); % Air
54
55
56 Tout = Tcold + dT;
57 set(CANTERA.gas{WISE},'T',Tout,'P',Phot*(1-ploss),'Y',massFractions(CANTERA.gas{WISE_hotin}));
58
59 if (flag==9 || flag==99) && CANTERA.DATA==1
60    Hhot = Xhot * enthalpy_mass(CANTERA.gas{WISE});
61    Shot = Xhot * entropy_mass(CANTERA.gas{WISE});
62    dHhot = Hhot - Xhot * enthalpy_mass(CANTERA.gas{WISE_hotin});
63    Z = [Xhot;Thot;Phot/oneatm;WISE_hotin; Xhot;Tout;Phot*(1-ploss)/oneatm;WISE;Hhot;Shot;dT;dHhot;0;0;0];
64    assignin('base',['intercooler_',num2str(WISE)],Z);
65 end
66
67 sys=[Tout];
68 %end %mdlOutputs
69
70
71 function sys=mdlGetTimeOfNextVarHit(t,x,u)
72 sampleTime = 1;    % Example, set the next hit to be one second later.
73 sys = t + sampleTime;
74 %end %mdlGetTimeOfNextVarHit
75
76 function sys=mdlTerminate(t,x,u,flag,dT,ploss,WISE)
77 % global CANTERA
78 % if CANTERA.DATA==1
79 %    Xhot=u(1); Thot=u(2); Phot=u(3)*oneatm; WISE_hotin=u(4);
80 % %    Xcold=u(5); Tcold=u(6); Pcold=u(7)*oneatm; WISE_coldin=u(8);
81 %    z=mdlOutputs(t,x,u,flag,dT,ploss,WISE);
82 %    Tout = z(1);
83 %
84 %    Hhot = Xhot * enthalpy_mass(CANTERA.gas{WISE});
85 %    Shot = Xhot * entropy_mass(CANTERA.gas{WISE});
86 %    dHhot = Hhot - Xhot * enthalpy_mass(CANTERA.gas{WISE_hotin});
87 %    Z = [Xhot;Thot;Phot/oneatm;WISE_hotin; Xhot;Tout;Phot*(1-ploss)/oneatm;WISE; Hhot;Shot;dT;dHhot;0;0;0];
88 %    assignin('base',['Intercooler_',num2str(WISE)],Z);
89 % end%if
```

```
90 sys = [];
91 %end %mdlTerminate
92
```

D.15. Flow Joiner Valve

/Users/winstonburbank/Documents/Matlab Work/PHD Library/sfun_Joiner.m 1 of 2

```
 1
 2 function [sys,x0,str,ts] = sfun_Joiner(t,x,u,flag,WISE)
 3 global CANTERA
 4 switch flag,
 5   case 0,   [sys,x0,str,ts]=mdlInitializeSizes(8,1);
 6             CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
 7             set(CANTERA.gas{WISE},'T',800,'P',1.5*oneatm,'X','CO2:0.1, H2O:0.18, N2:0.71, ↙
AR:0.01');
 8   case 1,    sys=mdlDerivatives(t,x,u);
 9   case 2,    sys=mdlUpdate(t,x,u);
10   case 3,    sys=mdlOutputs(t,x,u,WISE);
11   case 10,   sys=mdlOutputs(t,x,u,WISE);    x0=0; str=0; ts=0;
12   case 4,    sys=mdlGetTimeOfNextVarHit(t,x,u);
13   case 9,    sys=mdlTerminate(t,x,u);
14    otherwise,    error(['Unhandled flag = ',num2str(flag)]);
15 end %switch
16 %end %sfun
17
18 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
19 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES: end,
20 sizes = simsizes;
21 sizes.NumContStates  = 0;
22 sizes.NumDiscStates  = 0;
23 sizes.NumOutputs     = out;
24 sizes.NumInputs      = in;
25 sizes.DirFeedthrough = 1;   % u is needed for the calc of outputs=1
26 sizes.NumSampleTimes = 1;   % at least one sample time is needed
27 sys = simsizes(sizes);
28 x0  = []; % initialize the initial conditions
29 str = []; % str is always an empty matrix
30 ts  = [0 0]; % initialize the array of sample times
31 %end %mdlInitializeSizes
32
33 function sys=mdlDerivatives(t,x,u)
34 sys = [];
35 %end %mdlDerivatives
36
37 function sys=mdlUpdate(t,x,u)
38 sys = [];
39 %end %mdlUpdate
40
41
42 function [sys]=mdlOutputs(t,x,u,WISE)
43 global CANTERA
44 %%Inputs%%
45 X1=u(1); T1=u(2); P1=u(3); WISEin1=u(4);
46 X2=u(5); T2=u(6); P2=u(7); WISEin2=u(8);
```

```
47
48 if T1<CANTERA.LIMIT.T(1), T1=CANTERA.LIMIT.T(1); disp('T Error into Joiner'), end
49 if T2<CANTERA.LIMIT.T(1), T2=CANTERA.LIMIT.T(1); disp('T Error into Joiner'), end
50
51 Mf1= X1*massFractions(CANTERA.gas{WISEin1});
52 H1 = X1*enthalpy_mass(CANTERA.gas{WISEin1});
53
54 Mf2= X2*massFractions(CANTERA.gas{WISEin2});
55 H2 = X2*enthalpy_mass(CANTERA.gas{WISEin2});
56
57 X3=X1+X2;
58 P3=min([P1,P2]);
59 set(CANTERA.gas{WISE},'H',(H1+H2)/X3,'P',P3*oneatm,'Y',(Mf1+Mf2)./X3);
60 T3=temperature(CANTERA.gas{WISE});
61
62 if T3<CANTERA.LIMIT.T(1), T3=CANTERA.LIMIT.T(1); disp('T Error Leaving Joiner'),↙
end
63
64 sys=[T3];
65 %end %mdlOutputs
66
67
68 function sys=mdlGetTimeOfNextVarHit(t,x,u)
69 sampleTime = 1;    % Example, set the next hit to be one second later.
70 sys = t + sampleTime;
71 %end %mdlGetTimeOfNextVarHit
72
73 function sys=mdlTerminate(t,x,u)
74 sys = [];
75 %end %mdlTerminate
76
```

D.16. Losses

/Users/winstonburbank/Documents/Matlab Work/PHD Library/sfun_Losses.m 1 of 2

```
 1
 2 function [sys,x0,str,ts] = sfun_Losses(t,x,u,flag,Ploss,Tloss,WISE)
 3 global CANTERA
 4 switch flag,
 5  case 0,   [sys,x0,str,ts]=mdlInitializeSizes(4,0);
 6          CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
 7          set(CANTERA.gas{WISE},'T',300,'P',1*oneatm,'X','O2:0.21, N2:0.78, AR:0.01');
 8  case 1,    sys=mdlDerivatives(t,x,u);
 9  case 2,    sys=mdlUpdate(t,x,u,Ploss,Tloss,WISE);
10  case 3,    sys=mdlOutputs(t,x,u,Ploss,Tloss,WISE);
11  case {10,30,99},   sys=mdlOutputs(t,x,u,Ploss,Tloss,WISE);  x0=[]; str=[]; ts=[];
12          sys = [u(1);u(2)*(1-Tloss);u(3)*(1-Ploss);WISE];
13  case 4,    sys=mdlGetTimeOfNextVarHit(t,x,u);
14  case 9,    sys=mdlTerminate(t,x,u);
15   otherwise,    error(['Unhandled flag = ',num2str(flag)]);
16 end %switch
17 %end %sfun
18
19 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
20 global CANTERA
21 sizes = simsizes;
22 sizes.NumContStates  = 0;
23 sizes.NumDiscStates  = 0;
24 sizes.NumOutputs     = out;
25 sizes.NumInputs      = in;
26 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs=1
27 sizes.NumSampleTimes = 1;  % at least one sample time is needed
28 sys = simsizes(sizes);
29 x0  = [];  % initialize the initial conditions
30 str = [];  % str is always an empty matrix
31 ts  = [0 0];  % initialize the array of sample times
32 %end %mdlInitializeSizes
33
34 function sys=mdlDerivatives(t,x,u)
35 sys = [];
36 %end %mdlDerivatives
37
38 function sys=mdlUpdate(t,x,u,Ploss,Tloss,WISE)
39 sys = [];
40 %end %mdlUpdate
41
42
43 function [sys]=mdlOutputs(t,x,u,Ploss,Tloss,WISE)
44 global CANTERA
45 X=u(1); T=u(2); P=u(3); W=u(4);
46 set(CANTERA.gas{WISE},'T',T*(1-Tloss),'P',P*oneatm*(1-Ploss),'Y',massFractions↙
(CANTERA.gas{W}));
```

```
47 sys=[];
48 %end %mdlOutputs
49
50
51 function sys=mdlGetTimeOfNextVarHit(t,x,u)
52 sampleTime = 1;    %  Example, set the next hit to be one second later.
53 sys = t + sampleTime;
54 %end %mdlGetTimeOfNextVarHit
55
56 function sys=mdlTerminate(t,x,u)
57 sys = [];
58 %end %mdlTerminate
59
```

D.17. Properties (Thermal Probe)

/Users/winstonburbank/Documents/Matlab Work/PHD Libr.../sfun_properties.m 1 of 2

```
 1 function [sys,x0,str,ts] = sfun_properties(t,x,u,flag)
 2 global CANTERA
 3 switch flag,
 4   case 0,   [sys,x0,str,ts]=mdlInitializeSizes(4,2+3+CANTERA.NSP);
 5   %case 1,   sys=mdlDerivatives(t,x,u);
 6   %case 2,   sys=mdlUpdate(t,x,u);
 7   case 3,    sys=mdlOutputs(t,u);
 8   %case 10,   sys=mdlOutputs(t,x,u,n_Fuel,FC_Volt);  x0=0; str=0; ts=0;
 9   %case 4,   sys=mdlGetTimeOfNextVarHit(t,x,u);
10   %case 9,   sys=mdlTerminate(t,x,u);
11   case {1,2,10,4,9}, sys=[];
12    otherwise,   error(['Unhandled flag = ',num2str(flag)]);
13 end %switch
14 %end %sfun_properties
15
16 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
17 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
18 sizes = simsizes;
19 sizes.NumContStates  = 0;
20 sizes.NumDiscStates  = 0;
21 sizes.NumOutputs     = out;
22 sizes.NumInputs      = in;
23 sizes.DirFeedthrough = 1;   % u is needed for the calc of outputs
24 sizes.NumSampleTimes = 1;   % at least one sample time is needed
25 sys = simsizes(sizes);
26 x0  = []; % initialize the initial conditions
27 str = []; % str is always an empty matrix
28 ts  = [0 0]; % initialize the array of sample times
29 %end %mdlInitializeSizes
30
31 function [sys] = mdlOutputs(t,u)
32 global CANTERA;
33 X=u(1); T1=u(2); P1=u(3); WISEin=u(4);
34 T=temperature(CANTERA.gas{WISEin}); P=pressure(CANTERA.gas{WISEin})/oneatm;
35
36 if t~=0 && CANTERA.ERROR==1
37    if abs(T-T1)>10^-1%CANTERA.ROUNDING,
38       disp(['At time ',num2str(t),' Error in Temp from WISE ',num2str(WISEin),' T1= ',↙
num2str(T1),' Tgas= ',num2str(T)]);
39    end
40
41    if abs(P-P1)>10^-1%CANTERA.ROUNDING,
42       disp(['At time ',num2str(t),' Error in Pres from WISE ',num2str(WISEin),' P1= ',↙
num2str(P1),' Pgas= ',num2str(P)]);
43    end
44 end %if
45 S = X * entropy_mass(CANTERA.gas{WISEin}) / 1000;   %kW/K
```

```
46 H = X * enthalpy_mass(CANTERA.gas{WISEin}) / 1000;  %kW
47
48 set(CANTERA.gas{WISEin},'T',CANTERA.REF.T,'P',CANTERA.REF.P);
49 Ho= X * enthalpy_mass(CANTERA.gas{WISEin}) / 1000;  %kW
50 Hf= X * CANTERA.HV' * massFractions(CANTERA.gas{WISEin}) / 1; %kW
51 E = (H-Ho-Hf); %kW
52 set(CANTERA.gas{WISEin},'T',T,'P',P*oneatm);
53
54 Mf=X.*massFractions(CANTERA.gas{WISEin});
55
56 sys=[T;P;E;S;H;Mf];
57 %end %mdlOutputs
58
59
```

D.18. Flow Splitter Valve

/Users/winstonburbank/Documents/Matlab Work/PHD Library/sfun_Splitter.m 1 of 2

```
 1
 2 function [sys,x0,str,ts] = sfun_Splitter(t,x,u,flag,WISE1,WISE2)
 3 global CANTERA
 4 switch flag,
 5  case 0,  [sys,x0,str,ts]=mdlInitializeSizes(4,0);
 6          CANTERA.gas{WISE1} = importPhase('Basic.cti',CANTERA.GAS);
 7          CANTERA.gas{WISE2} = importPhase('Basic.cti',CANTERA.GAS);
 8          set(CANTERA.gas{WISE1},'T',800,'P',1.5*oneatm,'X','CO2:0.1, H2O:0.18, N2:↙
0.71, AR:0.01');
 9          set(CANTERA.gas{WISE2},'T',800,'P',1.5*oneatm,'X','CO2:0.1, H2O:0.18, N2:↙
0.71, AR:0.01');
10  case 1,   sys=mdlDerivatives(t,x,u);
11  case 2,   sys=mdlUpdate(t,x,u);
12  case 3,   sys=mdlOutputs(t,x,u,WISE1,WISE2);
13  case 10,  sys=mdlOutputs(t,x,u,WISE1,WISE2);   x0=0; str=0; ts=0;
14  case 4,   sys=mdlGetTimeOfNextVarHit(t,x,u);
15  case 9,   sys=mdlTerminate(t,x,u);
16   otherwise,   error(['Unhandled flag = ',num2str(flag)]);
17 end %switch
18 %end %sfun
19
20 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
21 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
22 sizes = simsizes;
23 sizes.NumContStates  = 0;
24 sizes.NumDiscStates  = 0;
25 sizes.NumOutputs     = out;
26 sizes.NumInputs      = in;
27 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs=1
28 sizes.NumSampleTimes = 1;  % at least one sample time is needed
29 sys = simsizes(sizes);
30 x0  = []; % initialize the initial conditions
31 str = []; % str is always an empty matrix
32 ts  = [0 0]; % initialize the array of sample times
33 %end %mdlInitializeSizes
34
35 function sys=mdlDerivatives(t,x,u)
36 sys = [];
37 %end %mdlDerivatives
38
39 function sys=mdlUpdate(t,x,u)
40 sys = [];
41 %end %mdlUpdate
42
43
44 function [sys]=mdlOutputs(t,x,u,WISE1,WISE2)
45 global CANTERA
```

```
46 %%inputs%%
47 X=u(1); T=u(2); P=u(3); WISEin=u(4);
48
49 if T<CANTERA.TMIN, T=CANTERA.TMIN; disp('T Error into Splitter'), end
50
51 x=massFractions(CANTERA.gas{WISEin});
52 set(CANTERA.gas{WISE1},'T',T,'P',P*oneatm,'Y',x);
53 set(CANTERA.gas{WISE2},'T',T,'P',P*oneatm,'Y',x);
54 sys=[];
55 %end %mdlOutputs
56
57
58 function sys=mdlGetTimeOfNextVarHit(t,x,u)
59 sampleTime = 1;    % Example, set the next hit to be one second later.
60 sys = t + sampleTime;
61 %end %mdlGetTimeOfNextVarHit
62
63 function sys=mdlTerminate(t,x,u)
64 sys = [];
65 %end %mdlTerminate
66
```

D.19. Set Gas Temperature

/Users/winstonburbank/Documents/Matlab Work/PHD .../sfun_Temp_Define.m 1 of 1

```
 1 function [sys,x0,str,ts] = sfun_Temp_Define(t,x,u,flag,T,WISE)
 2 global CANTERA
 3 switch flag,
 4  case 0,   [sys,x0,str,ts]=mdlInitializeSizes(4,0);
 5          CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
 6          set(CANTERA.gas{WISE},'T',500,'P',1*oneatm,'X','O2:0.21, N2:0.78, AR:0.01');
 7  case 3,   sys=mdlOutputs(t,u,T,WISE);
 8  case {1,2,10,4,9}, sys=[];
 9   otherwise,   error(['Unhandled flag = ',num2str(flag)]);
10 end %switch
11 %end %sfun_properties
12
13 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
14 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
15 sizes = simsizes;
16 sizes.NumContStates  = 0;
17 sizes.NumDiscStates  = 0;
18 sizes.NumOutputs     = out;
19 sizes.NumInputs      = in;
20 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs
21 sizes.NumSampleTimes = 1;  % at least one sample time is needed
22 sys = simsizes(sizes);
23 x0  = []; % initialize the initial conditions
24 str = []; % str is always an empty matrix
25 ts  = [0 0]; % initialize the array of sample times
26 %end %mdlInitializeSizes
27
28 function [sys] = mdlOutputs(t,u,T,WISE)
29 global CANTERA;
30 X=u(1); T1=u(2); P1=u(3); WISEin=u(4);
31 set(CANTERA.gas{WISE},'T',T,'P',P1*oneatm,'Y',massFractions(CANTERA.gas{WISEin}));
32 sys=[];
33 %end %mdlOutputs
34
35
```

D.20. Turbine Lookup

/Users/winstonburbank/Documents/Matlab Work/PH.../sfun_Turbine_Lookup.m 1 of 6

```
 1 %sfun_Turbine_Lookup
 2
 3 function [sys,x0,str,ts] = sfun_Turbine_Lookup(t,x,u,flag, WISE, map, ShaftSpeed_DP,↙
MachSpeed_DP, Eff_DP, V_X, Phi_X, ShaftSpeed_X)
 4 global CANTERA
 5 switch flag,
 6  case 0,   [sys,x0,str,ts]=mdlInitializeSizes(5,5);
 7          CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
 8          set(CANTERA.gas{WISE},'T',500,'P',5*oneatm,'X','O2:0.21, N2:0.78, AR:0.01');
 9          CANTERA.update{WISE}=15*oneatm;
10  case 1,   sys=mdlDerivatives(t,x,u);
11  case 2,   sys=[];%mdlUpdate();
12  case 3,   sys=mdlOutputs(t,x,u,flag, WISE, map, ShaftSpeed_DP, MachSpeed_DP,↙
Eff_DP, V_X, Phi_X, ShaftSpeed_X);
13  case {10,30,99},  sys=mdlOutputs(t,x,u,flag, WISE, map, ShaftSpeed_DP,↙
MachSpeed_DP, Eff_DP, V_X, Phi_X, ShaftSpeed_X);   x0=0; str=0; ts=0;
14          x0=sys(3); str=sys(4); ts=sys(5);
15          sys=[u(1);sys(1);sys(2);WISE];
16  case 4,   sys=mdlGetTimeOfNextVarHit(t,x,u);
17  case 9,   sys=mdlOutputs(t,x,u,flag, WISE, map, ShaftSpeed_DP, MachSpeed_DP,↙
Eff_DP, V_X, Phi_X, ShaftSpeed_X);sys=[];
18 %     sys=mdlTerminate(t,x,u,flag, WISE, map, ShaftSpeed_DP, MachSpeed_DP,↙
Eff_DP, V_X, Phi_X, ShaftSpeed_X);
19   otherwise,   error(['Unhandled flag = ',num2str(flag)]);
20 end %switch
21 end %sfun
22
23 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
24 % global NUM_GASSES, if isempty(NUM_GASSES), GLOBAL_VALUES; end,
25 sizes = simsizes;
26 sizes.NumContStates  = 0;
27 sizes.NumDiscStates  = 0;
28 sizes.NumOutputs     = out;
29 sizes.NumInputs      = in;
30 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs=1
31 sizes.NumSampleTimes = 1;  % at least one sample time is needed
32 sys = simsizes(sizes);
33 x0  = []; % initialize the initial conditions
34 str = []; % str is always an empty matrix
35 ts  = [0 0]; % initialize the array of sample times
36 end %mdlInitializeSizes
37
38 function sys=mdlDerivatives(t,x,u)
39 sys = [];
40 end %mdlDerivatives
41
42 function sys=mdlUpdate()
```

```
43 sys = [];
44 end %mdlUpdate
45
46
47 function [sys]=mdlOutputs(t,x,u,flag, WISE, map, ShaftSpeed_DP, MachSpeed_DP,↙
Eff_DP, V_X, Phi_X, ShaftSpeed_X)
48 global CANTERA
49 %%Inputs%%
50 X1=u(1); T1=u(2); P1=u(3)*oneatm; WISEin=u(4); C_Work=u(5);
51 if P1==0, P1 = pressure(CANTERA.gas{WISEin}), end
52 if P1 < CANTERA.LIMIT.P(1), P1 = CANTERA.LIMIT.P(1), end
53 if P1 > CANTERA.LIMIT.P(2), P1 = CANTERA.LIMIT.P(2), end
54 h1 = enthalpy_mass(CANTERA.gas{WISEin});
55 s1 = entropy_mass(CANTERA.gas{WISEin});
56 Phi = X1 * sqrt(T1) / (P1/1000); % kg_K^.5 / (kPa_s)
57 % CANTERA.update{WISE} = min(CANTERA.update{WISE}, P1/1.75);
58 set(CANTERA.gas{WISE},'T',T1,'P',CANTERA.update{WISE},'Y',massFractions(CANTERA.↙
gas{WISEin}));
59 options = optimset('display','off');
60
61 % V = 1; MachSpeed=1; ShaftSpeed=1;%intialize values for nested function
62 if C_Work < 0 %Error: meaning compressor is producing work
63    if CANTERA.ERROR>=1, disp(['Error at Wise ',num2str(WISE),': C_Work = ', num2str↙
(C_Work)]), end % if
64    P2 = P1;  eff = 0.666;  MachSpeed = 0; V = 0; ShaftSpeed=0; Work = 0;
65 else
66    t_work = -C_Work / X1 * 1000; % Watts
67    gama = cp_mass(CANTERA.gas{WISE}) / cv_mass(CANTERA.gas{WISE});
68 %    P2g = CANTERA.update{WISE};
69 %    P3g = real((t_work*(gama-1)/(CANTERA.REF.R*T1*0.83)+1)^-((gama-1)/gama)↙
* P1);
70    CANTERA.update{WISE} = real((t_work*(gama-1)/(CANTERA.REF.R*T1*0.83)+1)^-↙
((gama-1)/gama) * P1);
71 %    options = optimset('TolFun',0.1,'TolX',0.0000001);
72
73 %    Z=0;  %Tells Nested Function to use constant eff
74 %    P2 = fsolve(@Turbine,CANTERA.update{WISE},options);
75 %    CANTERA.update{WISE}=P2;
76    if CANTERA.DP ~= 1
77 %       Z=0;
78 %       P2 = fsolve(@Turbine,CANTERA.update{WISE},options);
79 %       P2 = min( P1, max(P2, CANTERA.LIMIT.P(1)));
80 %       CANTERA.update{WISE}=P2;
81       Z=1;   %Tells nested function to use true eff
82       P2 = fsolve(@Turbine,CANTERA.update{WISE},options);
83       P2 = min( P1, max(P2, CANTERA.LIMIT.P(1)));
84 %       CANTERA.update{WISE}=P2;
```

```
 85    else % CANTERA.DP == 1
 86        Z=0;
 87        P2 = fsolve(@Turbine,CANTERA.update{WISE},options);
 88        CANTERA.update{WISE}=P2;
 89        ER = P1/P2;
 90        V = 0.69;
 91        ShaftSpeed = ShaftSpeed_DP;
 92
 93        if flag==9 || flag==99
 94 %          speeds = interp1(map.ER, map.SPEEDS',ER);
 95 %          [b, m, n] = unique(speeds);
 96 %          Phi_map = interp1(speeds(m),map.PHI(m),MachSpeed_DP);
 97        if MachSpeed_DP>=min(map.speeddata(:,3)) && MachSpeed_DP<=max(map.↙
speeddata(:,3))
 98            k = interp1(map.speedvalues, map.k, MachSpeed_DP,'spline','extrap');
 99        else
100            k = interp1(map.speedvalues, map.k, MachSpeed_DP,'linear','extrap');
101        end
102           Phi_map = (1 - exp(-k * (ER-1)));
103           Phi_X = Phi / Phi_map;
104           MachSpeed = MachSpeed_DP;
105           ShaftSpeed_X = ShaftSpeed_DP / (MachSpeed_DP * sqrt(T1));
106
107           V_X = 0.69 / (ShaftSpeed_DP / sqrt(2*(h1-h2s)) );
108
109           disp(['For WISE # ',num2str(WISE)]);
110           disp(['PHI_X should equal ',num2str(Phi_X)]);
111           disp(['V_X should equal ',num2str(V_X)]);
112           disp(['ShaftSpeed_X should equal ',num2str(ShaftSpeed_X)]);
113           disp(' ');
114           assignin('base',['DP_WISE_',num2str(WISE)],[Phi_X,V_X,ShaftSpeed_X]);
115        end % if flag
116    end
117 %    C_Work;
118     Work = work*X1/1000;
119     if abs(work-t_work)>1
120        if CANTERA.ERROR>=1, disp(['Error work+c_work= ',num2str(work+t_work),' :↙
Wise= ',num2str(WISE)]); end %if
121 %        turbine_lookup_work_error = work-t_work,
122 %        work = t_work;
123     end
124
125     h2 = h1 + work;
126
127     P2 = min( P1, max(P2, CANTERA.LIMIT.P(1)));
128     setState_HP(CANTERA.gas{WISE},[h2,P2]);
129 end
```

```
130 T2 = temperature(CANTERA.gas{WISE});
131
132    function x = Turbine(P2g)
133       P2g = min( P1, max(P2g, 3*CANTERA.LIMIT.P(1)));
134 % pressure2guess = P2g/oneatm,
135       set(CANTERA.gas{WISE},'S',s1,'P',P2g);
136       h2s=enthalpy_mass(CANTERA.gas{WISE});
137       ERg = P1/P2g;
138
139       if CANTERA.DP==1 , eff = Eff_DP;  %V = (ShaftSpeed)/sqrt(2*(h1-h2s));
140       elseif Z==0, eff=Eff_DP * 0.95;
141       else
142 %          MachSpeed = interp2(map.ER,map.PHI,map.SPEEDS,ERg,Phi/Phi_X,'linear');
143          k = real(-log(1-Phi/Phi_X) / (ERg-1));
144          if k>=min(map.k) && k<=max(map.k)
145             MachSpeed = interp1(map.k, map.speedvalues, k,'spline','extrap');
146          else
147             MachSpeed = interp1(map.k, map.speedvalues, k,'linear','extrap');
148          end
149 %          MachSpeed = interp1(map.k, map.speedvalues, k,'linear','extrap');
150          if isnan(MachSpeed) || MachSpeed <0
151 %             if (Phi/Phi_X > max(map.PHI) || ERg<=1), MachSpeed=1, else↙
MachSpeed=120,  end
152             if (MachSpeed<0), MachSpeed=0; else MachSpeed=120,  end
153          end %if
154          ShaftSpeed = MachSpeed * sqrt(T1) * ShaftSpeed_X;
155          d = sqrt(2*(h1-h2s));
156          if d<=0,
157             d=0.00001,
158          end
159          V = (ShaftSpeed) / d * V_X;
160          eff = polyval(map.nor_poly_eff, V/0.69) * Eff_DP;
161          if eff<0.20 || eff>1 || ~isreal(eff) || isnan(eff),
162             if CANTERA.ERROR>=3, disp(['Error at Wise ',num2str(WISE),': eff = ',↙
num2str(eff),'  V*V_X = ',num2str(V*V_X),'  V = ',num2str(V)]), end % if
163             eff=0.20;
164          end
165       end
166
167       if ERg<1, work=(h2s-h1)/eff; if CANTERA.ERROR>=3, disp(['Error at Wise ',↙
num2str(WISE),'  Er < 1 !!']); end %Compressor
168       else     work=(h2s-h1)*eff; %Turbine
169       end %if
170       x = work - t_work;
171
172 %       disp(['P2g = ',num2str(P2g),'  T2g = ',num2str(temperature(CANTERA.gas↙
{WISE})),'  x = ',num2str(x)]);
```

```
173    end %Turbine
174
175 if (flag == 9 || flag==99) % terminate
176    if CANTERA.PLOT==1
177       ER = P1/P2;
178       figure(WISE), hold off
179       plot(map.speeddata(:,1),map.speeddata(:,2).* Phi_X,'g.')
180 %        contour(map.ER, map.PHI .* Phi_X, map.SPEEDS, [map.speedvalues]);↙
colorbar,
181       hold on, grid on
182       if MachSpeed>=min(map.speeddata(:,3)) && MachSpeed<=max(map.↙
speeddata(:,3))
183          k = interp1(map.speedvalues, map.k, MachSpeed,'spline','extrap');
184       else
185          k = interp1(map.speedvalues, map.k, MachSpeed,'linear','extrap');
186       end
187       plot(map.ER, (1 - exp(-k.*(map.ER-1))) * Phi_X,'k-', LineWidth',2);
188       plot(ER,Phi,'ro'),
189       title(sprintf(['eff = ',num2str(eff),'     PHI = ',num2str(Phi),'     ER = ',num2str↙
(ER),...
190          '\n MachSpeed = ',num2str(MachSpeed),'  ShaftSpeed = ',num2str↙
(ShaftSpeed),'   Work = ',num2str(Work)]))
191       xlabel('ER'),ylabel('Phi')
192    end %if
193    if CANTERA.DATA==1
194       H = X1 * enthalpy_mass(CANTERA.gas{WISE});
195       S = X1 * entropy_mass(CANTERA.gas{WISE});
196
197       Z = [X1;T1;P1/oneatm;WISEin; X1;T2;P2/oneatm;WISE; H;S;eff;Work;↙
ShaftSpeed;V;0];
198    assignin('base',['Turbine_',num2str(WISE)],Z);
199    end %if
200 end % flag ==9
201
202 sys=[T2;P2/oneatm;eff;ShaftSpeed;V];
203 end %mdlOutputs
204
205 function sys=mdlGetTimeOfNextVarHit(t,x,u)
206 sampleTime = 1;    %  Example, set the next hit to be one second later.
207 sys = t + sampleTime;
208 end %mdlGetTimeOfNextVarHit
209
210 % function sys=mdlTerminate(t,x,u,flag, WISE, map, ShaftSpeed_DP,↙
MachSpeed_DP, Eff_DP, V_X, Phi_X, ShaftSpeed_X)
211 % sys = [];
212 % end %mdlTerminate
213
```

D.21. Variable Inlet Vane Turbine Lookup

/Users/winstonburbank/Documents/Matlab Work/PHD Li.../sfun_Var_Turbine.m 1 of 5

```
 1 %sfun_Var_Turbine
 2 % inputs: XTPW P2 Nozzel
 3 % outputs: T Work X eff V ShaftSpeed
 4
 5 function [sys,x0,str,ts] = sfun_Var_Turbine(t,x,u,flag,WISE, map, ShaftSpeed_DP,↙
MachSpeed_DP, Eff_DP, V_X, Phi_X, ShaftSpeed_X, Constraint)
 6 global CANTERA
 7 % ShaftSpeed_DP = 100;
 8 switch flag,
 9  case 0,  [sys,x0,str,ts]=mdlInitializeSizes(6,6);
10          CANTERA.gas{WISE} = importPhase('Basic.cti',CANTERA.GAS);
11          set(CANTERA.gas{WISE},'T',500,'P',5*oneatm,'X','O2:0.21, N2:0.78, AR:0.01');
12  case 1,  sys=mdlDerivatives(t,x,u);
13  case 2,  sys=mdlUpdate(t,x,u);
14  case 3,  %disp(['Compressor: t=',num2str(t),' u= ',num2str(u),' WISE= ',num2str↙
(WISE)]),
15          sys=mdlOutputs(t,x,u,flag,WISE, map, ShaftSpeed_DP, MachSpeed_DP,↙
Eff_DP, V_X, Phi_X, ShaftSpeed_X, Constraint);
16  case {10,30,99},  sys=mdlOutputs(t,x,u,flag,WISE, map, ShaftSpeed_DP,↙
MachSpeed_DP, Eff_DP, V_X, Phi_X, ShaftSpeed_X, Constraint);
17          x0=sys(2); str=sys(3); ts=sys(4:6);
18          sys=[u(1);sys(1);u(5);WISE];
19  case 4,  sys=mdlGetTimeOfNextVarHit(t,x,u);
20  case 9,  sys=mdlOutputs(t,x,u,flag,WISE, map, ShaftSpeed_DP, MachSpeed_DP,↙
Eff_DP, V_X, Phi_X, ShaftSpeed_X, Constraint);
21          sys=[];
22    %    sys=mdlTerminate(t,x,u,flag,WISE, map, ShaftSpeed_DP, MachSpeed_DP,↙
Eff_DP, V_X, Phi_X, ShaftSpeed_X, Constraint);
23   otherwise,  error(['Unhandled flag = ',num2str(flag)]);
24 end %switch
25 end %sfun
26
27 function [sys,x0,str,ts]=mdlInitializeSizes(in,out)
28 % global NUM_GASSES; if isempty(NUM_GASSES), GLOBAL_VALUES; end,
29 sizes = simsizes;
30 sizes.NumContStates  = 0;
31 sizes.NumDiscStates  = 0;
32 sizes.NumOutputs     = out;
33 sizes.NumInputs      = in;
34 sizes.DirFeedthrough = 1;  % u is needed for the calc of outputs=1
35 sizes.NumSampleTimes = 1;  % at least one sample time is needed
36 sys = simsizes(sizes);
37 x0  = []; % initialize the initial conditions
38 str = []; % str is always an empty matrix
39 ts  = [0 0]; % initialize the array of sample times
40 end %mdlInitializeSizes
41
```

```
 42 function sys=mdlDerivatives(t,x,u)
 43 sys = [];
 44 end %mdlDerivatives
 45
 46 function sys=mdlUpdate(t,x,u)
 47 sys = [];
 48 end %mdlUpdate
 49
 50
 51 function [sys]=mdlOutputs(t,x,u,flag,WISE, map, ShaftSpeed_DP, MachSpeed_DP,↙
Eff_DP, V_X, Phi_X, ShaftSpeed_X, Constraint)
 52 global CANTERA
 53 %%Inputs%%
 54 X1=u(1); T1=u(2); P1=u(3)*oneatm; WISEin=u(4); P2=u(5)*oneatm; Nozzel=u(6);
 55 Eff_Nozzel_depart = -0.2133*(Nozzel/100)^2 + 0.4267*Nozzel/100 + 0.7867;
 56 Phi = X1 * sqrt(T1) / (P1/1000); % kg_K^.5 / (kPa_s)
 57 if P1<=0, P1 = pressure(CANTERA.gas{WISEin}); end
 58 ER = P1 / P2;
 59 if ER < 1,
 60    if CANTERA.ERROR>=3, disp(['Var Turbine ER<1']); end
 61    eff = 0.666,  u(6)=eff;
 62    [sys,x0,str,ts] = sfun_Compressor(t,x,u,10,WISE);
 63    T2 = sys(1); work = sys(2); X_needed=0; V=0; ShaftSpeed=0;
 64 else % ER >=1
 65    h1 = enthalpy_mass(CANTERA.gas{WISEin});
 66    s1 = entropy_mass(CANTERA.gas{WISEin});
 67    set(CANTERA.gas{WISE},'S',s1,'P',P2,'Y',massFractions(CANTERA.gas{WISEin}));
 68    h2s=enthalpy_mass(CANTERA.gas{WISE});
 69    dhs = max( 0.00001, (h1-h2s) );
 70
 71    if CANTERA.DP == 1,
 72       MachSpeed = MachSpeed_DP;
 73       ShaftSpeed = ShaftSpeed_DP;
 74       eff = Eff_DP*Eff_Nozzel_depart;
 75       V = 1;
 76
 77 %       speeds = interp1(map.ER, map.SPEEDS',ER);
 78 %       [b, m, n] = unique(speeds);
 79 %       a=speeds(m); b=map.PHI(m);
 80 %       Phi_map = interp1(speeds(m),map.PHI(m),MachSpeed);
 81       if MachSpeed>=min(map.speeddata(:,3)) && MachSpeed<=max(map.↙
speeddata(:,3))
 82          k = interp1(map.speedvalues, map.k, MachSpeed,'spline','extrap');
 83       else
 84          k = interp1(map.speedvalues, map.k, MachSpeed,'linear','extrap');
 85       end
 86       Phi_map = (1 - exp(-k * (ER-1)));
```

```
 87      Phi_needed = Phi;
 88      Phi_X = Phi / Phi_map;
 89      ShaftSpeed_X = ShaftSpeed / (MachSpeed * sqrt(T1));
 90      V_X = 0.69 / (ShaftSpeed / sqrt(2*dhs));
 91
 92      X_needed = (Phi_map * Phi_X) * (P1/1000) / sqrt(T1); % kg_K^.5 / (kPa_s);
 93
 94    else %CANTERA.DP == 0
 95      switch Constraint
 96        case 1  % Constrained Shaft Speed
 97          ShaftSpeed = ShaftSpeed_DP;
 98        case 2  % Constrained Mach Speed
 99          MachSpeed = MachSpeed_DP;
100           ShaftSpeed = MachSpeed * sqrt(T1) * ShaftSpeed_X;
101         case 3  % Constrained Max Eff: U/Co = 0.69
102           ShaftSpeed = 0.69 * sqrt(2*dhs) / V_X;% * ShaftSpeed_X;
103           % FP Turbine constrained to best efficiency
104   %     % For Constant MachSpeed condition
105      end
106      MachSpeed = ShaftSpeed / sqrt(T1) / ShaftSpeed_X;
107
108 %      speeds = interp1(map.ER, map.SPEEDS',ER,'linear','extrap');
109 %      [b, m, n] = unique(speeds,'last');
110 %      Phi_needed = interp1(speeds(m),map.PHI(m),MachSpeed) * Phi_X *↙
Nozzel/100;
111      if MachSpeed>=min(map.speeddata(:,3)) && MachSpeed<=max(map.↙
speeddata(:,3))
112          k = interp1(map.speedvalues, map.k, MachSpeed,'spline','extrap');
113      else
114          k = interp1(map.speedvalues, map.k, MachSpeed,'linear','extrap');
115      end
116      Phi_needed = (1 - exp(-k * (ER-1))) * Phi_X * Nozzel/100;
117      X_needed = Phi_needed * (P1/1000) / sqrt(T1); % kg_K^.5 / (kPa_s)
118
119      if X_needed<0;
120        X_needed=X1,
121      end
122
123 %      V = (ShaftSpeed / ShaftSpeed_X) / sqrt(2*dhs) * V_X;
124      V = ShaftSpeed / sqrt(2*dhs) * V_X;
125      eff = polyval(map.nor_poly_eff, V/0.69) * Eff_DP * Eff_Nozzel_depart;
126      if eff<0.1 || isnan(eff),
127        if CANTERA.ERROR==1, disp(['Eff = ',num2str(eff),' Being set to 0.1']), end
128         eff = 0.1,
129      end
130    end
131
```

```
132     work=(h2s-h1)*eff; %Turbine
133     h2 = h1 + work;
134     setState_HP(CANTERA.gas{WISE},[h2,P2]);
135     T2 = temperature(CANTERA.gas{WISE});
136 end
137
138 Work = work * X1 / 1000;
139 if (flag == 9 || flag==99)
140     if CANTERA.DP==1
141         disp(['For WISE # ',num2str(WISE)]);
142         disp(['Phi_X should equal ',num2str(Phi_X)]);
143         disp(['V_X should equal ',num2str(V_X)]);
144         disp(['ShaftSpeed_X should equal ',num2str(ShaftSpeed_X)]);
145         disp(' ');
146         assignin('base',['DP_WISE_',num2str(WISE)],[Phi_X,V_X,ShaftSpeed_X,Work]); %↙
Changed Work from work/1000
147     end
148
149     if CANTERA.PLOT==1
150         figure(WISE), hold off
151         plot(map.speeddata(:,1),map.speeddata(:,2).* Phi_X,'g.')
152         hold on
153 %         contour(map.ER, map.PHI .* Phi_X, map.SPEEDS, [MachSpeed_DP,↙
MachSpeed_DP],'b-');
154 %         contour(map.ER, map.PHI .* (Phi_X * Nozzel/100), map.SPEEDS, [MachSpeed,↙
MachSpeed],'k-');
155         k = interp1(map.speedvalues, map.k, MachSpeed);
156 %         k_DP = interp1(map.speedvalues, map.k, MachSpeed_DP);
157 %         plot(map.ER, (1 - exp(-k_DP.*(map.ER-1))) * Phi_X,'b-');
158         plot(map.ER, (1 - exp(-k.*(map.ER-1))) * Phi_X,'b-');
159         plot(map.ER, (1 - exp(-k.*(map.ER-1))) * Phi_X * Nozzel/100,'k-','LineWidth',2);
160
161         plot(ER,Phi,'ro',ER,Phi_needed,'rx'),
162         legend('data','Unadjusted↙
MachSpeed','MachSpeed','actual','needed','location','southeast'),
163         axis('auto'), grid on
164         title(sprintf(['eff = ',num2str(eff),'    PHI = ',num2str(Phi),'    ER = ',num2str↙
(ER),...
165             '\n MachSpeed = ',num2str(MachSpeed),'  ShaftSpeed = ',num2str↙
(ShaftSpeed),'  Work = ',num2str(Work),...
166             '\n Nozzel = ',num2str(Nozzel),'  X needed = ',num2str(X_needed),'  X↙
actual = ',num2str(X1)]));
167         xlabel('ER'),ylabel('Phi'),
168 %         diff=X_needed - X1,
169     end %if
170     if CANTERA.DATA==1
171         H = X1 * enthalpy_mass(CANTERA.gas{WISE});
```

```
172       S = X1 * entropy_mass(CANTERA.gas{WISE});
173
174       Z = [X1;T1;P1/oneatm;WISEin; X1;T2;P2/oneatm;WISE; H;S;eff;Work;↙
ShaftSpeed;V;Nozzel;X_needed];
175       assignin('base',['Var_Turbine_',num2str(WISE)],Z);
176    end %if
177 end %if flag==9
178 sys=[T2;Work;X_needed;eff;V;ShaftSpeed];
179 end %mdlOutputs
180
181
182 function sys=mdlGetTimeOfNextVarHit(t,x,u)
183 sampleTime = 1;    %  Example, set the next hit to be one second later.
184 sys = t + sampleTime;
185 end %mdlGetTimeOfNextVarHit
186
187 % function sys=mdlTerminate(t,x,u,flag,WISE, map, ShaftSpeed_DP,↙
MachSpeed_DP, Eff_DP, V_X, Phi_X, ShaftSpeed_X, Constraint)
188 % sys = [];
189 % end %mdlTerminate
190
```

D.22. Matlab Code to Run SOFC-GT Hybrid Engine


```
1327 %% Run Model
1328 function Error = Wolf_Engine_Run(X)
1329 %    disp(['TIT = ',num2str(Comb_Temp),' Nozzle = ',num2str(FPT_Nozzle),'
n_Fuel_FC = ',num2str(FC_n_Fuel),' X1g = ',num2str(X(1)),' LP Speedg = ',num2str(X(2)),'
HP Speedg = ',num2str(X(3)),' THXg = ',num2str(X(4))]);
1330
1331    X1g=abs(X(1)); LP_Speedg=abs(X(2)); HP_Speedg=abs(X(3)); THXg=abs(X(4));
1332
1333    % Determine Pressure out from FP_Turbine
1334    P10=1/(1-Engine.DP.Heat_Exchanger.Ploss_hot)/(1-Engine.DP.OutletFilter.
Ploss);
1335
1336    %[sys,x0,str,ts] = sfun_Gas_Input(t,x,u,flag,Gas_Type,Gas_Flow,WISE,Mf_T_P)
1337    [XTPW1,x0,str,ts] = sfun_Gas_Input(1,[],[],flag,1,X1g,Engine.DP.AirInlet.WISE,
zeros(1,CANTERA.NSP+2));
1338
1339    X=Engine.DP.InletFilter; % Inlet Filter Pressure Loss
1340    %[sys,x0,str,ts] = sfun_Losses(t,x,u,flag,Ploss,Tloss,WISE)
1341    [XTPW2,x0,str,ts] = sfun_Losses(1,[],XTPW1,flag,X.Ploss,X.Tloss,X.WISE);
1342
1343    X=Engine.DP.LPC; % LP Compressor
1344    %[sys,x0,str,ts] = sfun_Compressor_Lookup(t,x,u,flag, WISE, map,PR_DP,
MachSpeed_DP, SurgeMargin_DP, ShaftSpeed_X, Phi_X, PR_X)
1345    [XTPW3,LPC_Work,LPC_n,ts] = sfun_Compressor_Lookup(1,[],[XTPW2;
LP_Speedg],flag,X.WISE,X.map,X.PR_DP,X.MachSpeed_DP,X.SurgeMargin_DP,X.Eff_DP,X
ShaftSpeed_X,X.Phi_X,X.PR_X,X.Eff_X);
1346
1347
1348    X=Engine.DP.InterCooler; % Create Air Flow for Intercooler
1349    %[sys,x0,str,ts] = sfun_Gas_Input(t,x,u,flag,Gas_Type,Gas_Flow,WISE,Mf_T_P)
1350    [XTPWIntercoolerAir,x0,str,ts] = sfun_Gas_Input(1,[],[],flag,1,1,X.WISE_Air,zeros
(1,CANTERA.NSP+2));
1351    %[sys,x0,str,ts] = sfun_Intercooler(t,x,u,flag,dT,ploss,WISE)
1352    [XTPW4,x0,str,ts] = sfun_Intercooler(1,[],[XTPW3;XTPWIntercoolerAir],flag,X.dT,
X.Ploss_hot,X.WISE_hot);
1353
1354    X=Engine.DP.HPC; % HP Compressor
1355    %[sys,x0,str,ts] = sfun_Compressor_Lookup(t,x,u,flag, WISE, map, PR_DP,
MachSpeed_DP, SurgeMargin_DP, ShaftSpeed_X, Phi_X, PR_X)
1356    [XTPW5,HPC_Work,HPC_n,ts] = sfun_Compressor_Lookup(1,[],[XTPW4;
HP_Speedg],flag,X.WISE,X.map,X.PR_DP,X.MachSpeed_DP,X.SurgeMargin_DP,X.Eff_DP,X.
ShaftSpeed_X,X.Phi_X,X.PR_X,X.Eff_X);
1357
1358    X=Engine.DP.Heat_Exchanger; % Set Gas to guessed THXg outlet temp with
pressure drop
1359    % [sys,x0,str,ts] = sfun_Gas_Set(t,x,u,flag,WISE)P6 = XTPW5(3)*(1-PlossHXcold);
1360    [XTPW6g,x0,str,ts] = sfun_Gas_Set([],[],[XTPW5;THXg;XTPW5(3)*(1-X.
```

```
Ploss_cold)],flag,X.WISE_cold);
1361
1362    if FUELCELL == 0
1363        X=Engine.DP.Combustor;  % Create Fuel Flow for combustor
1364        %[sys,x0,str,ts] = sfun_Gas_Input(t,x,u,flag,Gas_Type,Gas_Flow,WISE,Mf_T_P)
1365        [XTPW30,x0,str,ts] = sfun_Gas_Input([],[],[],flag,2,CANTERA.update{X.WISE},X.↙
WISE_Fuel1,zeros(1,CANTERA.NSP+2));
1366        P31 = XTPW6g(3)*1.05;  T31 = 300;
1367        % [sys,x0,str,ts] = sfun_Gas_Set(t,x,u,flag,WISE)
1368        [XTPW31,x0,str,ts] = sfun_Gas_Set([],[],[XTPW30;T31;P31],flag,X.WISE_Fuel2);
1369        %  Combust Inlet air and fuel
1370        % [sys,x0,str,ts] = sfun_Combustor(t,x,u,flag,Tout, hloss, Ploss, WISE)
1371        [XTPW7,XCombFuel,str,ts] = sfun_Combustor([],[],[XTPW6g;XTPW31],flag,↙
Comb_Temp,X.Hloss,X.Ploss,X.WISE);
1372
1373    elseif FUELCELL == 1
1374        X=Engine.DP.FC;
1375        Y=Engine.DP.FC.WISE;
1376        %  Create Fuel flow for Fuel Cell
1377        %[sys,x0,str,ts] = sfun_Gas_Input(t,x,u,flag,Gas_Type,Gas_Flow,WISE,Mf_T_P)
1378        [XTPW44,x0,str,ts] = sfun_Gas_Input([],[],[],flag,2,CANTERA.update{Y.Fuel2},Y.↙
Fuel1,zeros(1,CANTERA.NSP+2));
1379        P45 = XTPW6g(3)*1.05;  T45 = 300;
1380        % [sys,x0,str,ts] = sfun_Gas_Set(t,x,u,flag,WISE)
1381        [XTPW45,x0,str,ts] = sfun_Gas_Set([],[],[XTPW44;T45;P45],flag,Engine.DP.FC.↙
WISE.Fuel2);
1382        %          sys=mdlOutputs(t,x,u,flag, FC_Type, n_Fuel, rec, Ploss_anode,↙
Ploss_cathode, Ploss_combustor, Ploss_HX,  FC_Volt, HX_Eff, CellArea, CellTemp,↙
FCExitTemp,...
1383        %  WISEAirHX, WISEAirFC, WISEPComb, WISEPHX, WISEFuelReform,
1384        %  WISEFuelFC, WISErec1, WISErec2);
1385        [sys,x0,str,ts]=sfun_Fuel_Cell([],[],[XTPW45;XTPW6g],flag, X.FC_Type,↙
FC_n_Fuel, X.rec, X.Ploss_anode,...
1386            X.Ploss_cathode, X.Ploss_combustor, X.Ploss_HX,  X.FC_Volt, X.HX_Eff, X.↙
CellArea, X.CellTemp, X.FCExitTemp,...
1387            Y.AirHX, Y.AirFC, Y.PComb, Y.PHX,Y.FuelReform, Y.FuelFC, Y.rec1, Y.rec2);
1388        XTPW43 = sys(13:16)';
1389        XFCFuel = sys(33);
1390        CANTERA.update{Y.Fuel2}=XFCFuel;
1391 %        FC_Work = sys(34);
1392 %        Flow_Check = XTPW6g(1) + XFCFuel - XTPW43(1);
1393
1394        if isfield(Engine.DP,'Transient')
1395        XTPW43 = [XTPW43(1)*(1+Engine.DP.Transient.FCdX); XTPW43(2); XTPW43(3)↙
*(1+Engine.DP.Transient.FCdP); XTPW43(4)];
1396        end
1397
```

```
1398        X=Engine.DP.Combustor; % Create Fuel flow for Combustor
1399        %[sys,x0,str,ts] = sfun_Gas_Input(t,x,u,flag,Gas_Type,Gas_Flow,WISE,Mf_T_P)
1400        [XTPW30,x0,str,ts] = sfun_Gas_Input([],[],[],flag,2,CANTERA.update{X.WISE},X.↙
WISE_Fuel1,zeros(1,CANTERA.NSP+2));
1401        P31 = XTPW43(3)*1.05;  T31 = 300;
1402        % [sys,x0,str,ts] = sfun_Gas_Set(t,x,u,flag,WISE)
1403        [XTPW31,x0,str,ts] = sfun_Gas_Set([],[],[XTPW30;T31;P31],flag,X.WISE_Fuel2);
1404        % [sys,x0,str,ts] = sfun_Combustor(t,x,u,flag,Tout, hloss, Ploss, WISE)
1405        [XTPW7,XCombFuel,str,ts] = sfun_Combustor([],[],[XTPW43;XTPW31],flag,↙
Comb_Temp,X.Hloss,X.Ploss,X.WISE);
1406    end
1407    CANTERA.update{Engine.DP.Combustor.WISE}=XCombFuel;
1408
1409    X=Engine.DP.HPT;  HPT_Work = HPC_Work/(1-Engine.DP.HPT.MechLoss);
1410    % [sys,x0,str,ts] = sfun_Turbine_Lookup(t,x,u,flag, WISE, map, ShaftSpeed_DP,↙
MachSpeed_DP, Eff_DP, V_X, Phi_X, ShaftSpeed_X)
1411    [XTPW8,HPT_eff,HPT_ShaftSpeed,HPT_V] = sfun_Turbine_Lookup([],[],[XTPW7;↙
HPT_Work],flag,X.WISE,X.map,X.ShaftSpeed_DP,X.MachSpeed_DP,X.Eff_DP,X.V_X,X.↙
Phi_X,X.ShaftSpeed_X);
1412
1413    X=Engine.DP.LPT;  LPT_Work = LPC_Work/(1-Engine.DP.LPT.MechLoss);
1414    % [sys,x0,str,ts] = sfun_Turbine_Lookup(t,x,u,flag, WISE, map, ShaftSpeed_DP,↙
MachSpeed_DP, Eff_DP, V_X, Phi_X, ShaftSpeed_X)
1415    [XTPW9,LPT_eff,LPT_ShaftSpeed,LPT_V] = sfun_Turbine_Lookup([],[],[XTPW8;↙
LPT_Work],flag,X.WISE,X.map,X.ShaftSpeed_DP,X.MachSpeed_DP,X.Eff_DP,X.V_X,X.Phi_X,↙
X.ShaftSpeed_X);
1416
1417    X=Engine.DP.FPT;
1418    % [sys,x0,str,ts] = sfun_Var_Turbine(t,x,u,flag,WISE, map, ShaftSpeed_DP,↙
MachSpeed_DP, Eff_DP, V_X, Phi_X, ShaftSpeed_X, Constraint)
1419    [XTPW10,FPT_Work,X_needed,FPT__eff_V_ShaftSpeed] = sfun_Var_Turbine([],[],↙
[XTPW9;P10;FPT_Nozzle],flag,X.WISE,X.map,X.ShaftSpeed_DP,X.MachSpeed_DP,X.Eff_DP,↙
X.V_X,X.Phi_X,X.ShaftSpeed_X,X.Constraint);
1420
1421
1422    % [sys,x0,str,ts] = sfun_HeatExchanger(t,x,u,flag,n_HX,ploss_hot,ploss_cold,↙
WISE_hot,WISE_cold,TCo)
1423    X=Engine.DP.Heat_Exchanger;
1424    [XTPW11,XTPW6,str,ts] = sfun_HeatExchanger([],[],[XTPW10;XTPW5],flag,X.↙
n_HX,X.Ploss_hot,X.Ploss_cold,X.WISE_hot,X.WISE_cold,THXg);
1425
1426    %[sys,x0,str,ts] = sfun_Losses(t,x,u,flag,Ploss,Tloss,WISE)
1427    X=Engine.DP.OutletFilter;
1428    [XTPW12,x0,str,ts] = sfun_Losses(1,[],XTPW11,flag,X.Ploss,X.Tloss,X.WISE);
1429
1430 %    for e=1:13
1431 %       switch e
```

```
1432 %          case 1, errorcheck(XTPW1);
1433 %          case 2, errorcheck(XTPW2);
1434 %          case 3, errorcheck(XTPW3);
1435 %          case 4, errorcheck(XTPW4);
1436 %          case 5, errorcheck(XTPW5);
1437 %          case 6, errorcheck(XTPW6);
1438 %          case 7, errorcheck(XTPW7);
1439 %          case 8, errorcheck(XTPW8);
1440 %          case 9, errorcheck(XTPW9);
1441 %          case 10, errorcheck(XTPW10);
1442 %          case 11, errorcheck(XTPW11);
1443 %          case 12, errorcheck(XTPW12);
1444 %          case 13, if FUELCELL==1,  errorcheck(XTPW43); end
1445 %       end
1446 %    end
1447
1448     Error_X   = (X_needed - XTPW11(1))*1000;
1449     Error_LP  = (LPT_ShaftSpeed - LP_Speedg);
1450     Error_HP  = (HPT_ShaftSpeed - HP_Speedg);
1451     Error_THX = (XTPW6(2) - THXg);
1452
1453 %     disp(['X1g = ',num2str(X1g),'  X needed = ',num2str(X_needed),'  THXg = ',↙
num2str(THXg),'  THX actual = ',num2str(XTPW6(2))]);
1454 %     disp(['LP_Speedg = ',num2str(LP_Speedg),'  LPT_ShaftSpeed = ',num2str↙
(LPT_ShaftSpeed),'  HP_Speedg = ',num2str(HP_Speedg),'  HPT_ShaftSpeed = ',num2str↙
(HPT_ShaftSpeed)]);
1455 %     disp(' ');
1456
1457     Error=[Error_X; Error_LP; Error_HP; Error_THX]';
1458     end %Wolf_Engine_Run
1459
1460 %     function errorcheck(error)
1461 %        T=error(2); P=error(3); WISE=error(4);
1462 %        TC=temperature(CANTERA.gas{WISE}); PC=pressure(CANTERA.gas{WISE})↙
/oneatm;
1463 %        if abs(T-TC)>10^-1
1464 %           disp([' Error in Temp from WISE ',num2str(WISE),' T= ',num2str(T),'↙
Tcantera= ',num2str(TC)]);
1465 %        end
1466 %
1467 %        if abs(P-PC)>10^-1
1468 %           disp([' Error in Pres from WISE ',num2str(WISE),' P= ',num2str(P),'↙
Pcantera= ',num2str(PC)]);
1469 %        end
1470 %     end
```

D.23. Basic Gas Composition Linking to GRI30

/Users/winstonburbank/Documents/Matlab Work/PHD Library/Basic.cti 1 of 1

```
1 ##  Basic cti designed by Winston Burbank for use with the UAF model ##
2
3 ideal_gas(name = "basic8",
4      elements = " O  H  C  N  Ar ",
5      species = """gri30: CO  CO2  H2  H2O  N2  O2  AR  CH4 """,
6      #reactions = "all",
7      #kinetics = "GRI30",
8      initial_state = state(temperature = 800.0, pressure = OneAtm)    )
9
10
11 ideal_gas(name = "basic13",
12      elements = " O  H  C  N  Ar ",
13      species = """gri30: CO  CO2  H2  H2O  N2  O2  AR  CH4
14                    C2H6  C3H8  C  NO  NO2 """,
15      #reactions = "all",
16      #kinetics = "GRI30",
17      initial_state = state(temperature = 800.0,
18                    pressure = OneAtm)    )
19
```